SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

RAČUNALSTVO

Edouard Ivanjko, Mario Muštra

Zagreb, 2016.

Ovu skriptu posvećujemo svim ljudima željnih stalnog usavršavanja i napredovanja u životu.

Zahvala

Autori zahvaljuju svim nastavnicima i studentima čiji su komentari te dojave pogrešaka unaprijedili ovu skriptu.

Predgovor

Sadržaj ove skripte čine nastavne teme koje su uključene u predavanja i laboratorijske vježbe predmeta Računalstvo koji se predaje na prvoj godini zajedničkog studija PROMET, ITS i logistika te aeronautika na Fakultetu prometnih znanosti Sveučilišta u Zagrebu. Namjena predmeta Računalstvo je dati studentima uvod u područje računalnih znanosti te njihovu primjenu u sklopu tehničkog područja tehnologija prometa i transport. Nastavne teme predmeta Računalstvo uključuju:

- Ustroj i načela rada računala;
- Primjena aritmetičkih, trigonometrijskih i logičkih operatora u računalstvu, odnosno programiranju;
- Izrada pseudokôda;
- Izrada i provjera dijagrama toka;
- Uvod u programski jezik C♯.

Namjena ovog nastavnog materijala je da olakša studentima usvajanje gradiva koje im je objašnjeno na predavanjima i laboratorijskim vježbama. Sadržaj skripte je složen u slijedu koji studentu omogućuje usvajanje novog znanja korak po korak. Svaka cjelina, gdje je to bilo moguće uključiti, sadrži primjere koji pokazuju primjenu obrađenih tema. Namjera autora je također potaknuti studente na samostalan rad i usavršavanje u području primjene računala koje je danas nezaobilazno u svim poljima. Naročito, u poljima koja se bave upravljanjem velikih i složenih sustava kao što je to promet i transport.

Cjelokupni nastavni materijal ove skripte kao i primjeri implementirani u programskim paketima Raptor i Visual Studio dostupni su na osobnim web stranicama autora (fpz.unizg.hr/eivanjko/) te unutar sustava e-učenja Fakulteta prometnih znanosti e-Student i SRCE-a Merlin. U slučaju otkrića bilo kakvih pogrešaka ili novih informacija koje bi bilo dobro uključiti u ovu skriptu molimo da nam se javite na e-mail adrese edouard.ivanjko@fpz.hr i mario.mustra@fpz.hr

uz korištenje ključne riječi "[RAČ]" na početku naslova poruke.

Zagreb, veljača 2016.

A U T O R I

Sadržaj

Poglavlje 1

Uvod

Razvoj elektroničkih računala omogućio je njihovu široku primjenu. Računala su prisutna u svim područjima znanosti, primjene znanosti u industriji i svakodnevnom životu. Jedna od vrlo značajnih primjena je i područje tehnologije prometa i transport. Današnja računala omogućuju mjerenje opterećenja transportne mreže, provjeru ispravnosti transportne infrastrukture, optimiranje korištenja transportne mreže (kreiranje ruta za vozila, odabir moda transporta, kreiranje voznih redova), planiranje gradnje nove infrastrukture i sl.

U prošlih nekoliko desetljeća prisutno je povezivanje svih modova transporta u jednu cjelinu. Velike tvrtke za dostavu paketa danas koriste mogućnost dostave od vrata do vrata. Pri tome se za različite udaljenosti koriste različiti modovi transporta pa paket preuzima dostavljač koji dolazi pješice, biciklom ili manjim cestovnim vozilom. U lokalnom sabirnom centru paketi se razvrstavaju i prevoze većim cestovnim vozilom u glavni regionalni sabirni centar. Takvi centri često su povezani sa željezničkom mrežom ili imaju osiguranu poveznicu s obližnjim aerodromom, odnosno morskom ili riječnom lukom. Tako se paketi za dostavu na veće udaljenosti prevoze vlakovima, zrakoplovima i brodovima. Pri tome se optimizira cijena prijevoza i brzina dostave. Svaki paket ima mogućnost praćenja njegovog trenutnog položaja kao i procijenjenog vremena dostave paketa. Sve navedeno nije moguće bez primjene računala.

Jedna od najvažnijih primjena računala u prometu i transportu je u prikupljanju te obradi podataka o opterećenju transportne mreže. Skoro svaki današnji mobilni uređaj (pametni telefoni, vozila s ugrađenim sustavom praćenja, inteligentna osjetila s ugradbenim računalom radi lokalne obrade mjernih podataka) predstavlja osjetilo za mjerenje svog položaja i stanja svoje okoline. Svaki korisnik transportne mreže postaje mobilno osjetilo koje mjeri vlastito

stanje (položaj, brzinu, napredovanje po unaprijed isplaniranoj ruti putovanja i dr.) i stanje svoje okoline (kvalitetu zraka, temperaturu, položaj i vrstu horizontalne te vertikalne cestovne signalizacije, ispravnost ponašanja okolnih korisnika transportne mreže i dr.). Veliku važnost u suvremenim transportnim sustavima ima mogućnost razmjene podataka korištenjem žičane ili bežične telekomunikacijske mreže. Koriste se različiti protokoli razmjene podataka, a radi lakše primjene protokola razmjene i prikaza podataka razvijena je i pripadna programska podrška, odnosno mrežno sučelje koje danas znamo pod nazivom Internet. Pomoću Interneta moguće je obaviti prijenos različitih vrsta podataka (teksta, slika, audio i video sadržaja) u stvarnom vremenu.

Za različite obrade prikupljenih podataka koriste se algoritmi, odnosno konačan slijed dobro definiranih naredbi za ostvarenje zadatka. Riječ "algoritam" dolazi od latinskog prijevoda imena iranskog matematičara Al-Hvarizmija (puno ime Abu Džafar Muhamad ibn Musa-al-Hvarizmi) [1] koji se bavio trigonometrijom, astronomijom, zemljopisom i kartografijom, a smatra se ocem algebre. Algoritam je postupak koji za dano početno stanje završava u definiranom konačnom stanju. U sklopu tehnologije prometa i transporta vrlo važno mjesto zauzimaju algoritmi optimiziranja, algoritmi za prikupljanje, arhiviranje, obradu i pretraživanje velike količine podataka te algoritmi zasnovani na umjetnoj inteligenciji.

U ovoj skripti dan je uvod u znanja potrebna za kreiranje osnovne programske podrške koja je jezgra svakog sustava za upravljanje, nadzor i korištenje transportne mreže. Nakon što usvoji znanja opisana u ovoj skripti student će biti u mogućnosti razraditi i prilagoditi dani prometni problem na način da se njegovo rješenje može implementirati u obliku jednostavnog (tekstualnog) računalnog programa. Usvajanjem znanja u okviru predmeta nasljednika predmeta Računalstvo, student će biti u mogućnosti nadograditi svoje znanje u vidu dodavanja grafičkog korisničkog sučelja, rada s datotekama te dodavanja sučelja s ostvarenjem komunikacije s drugim programima korištenjem Interneta. Koraci u tom postupku uključuju:

1. Proučavanje i razradu problema;
2. Izradu popisa potrebnih varijabli;
3. Prilagodbu matematičkih izraza za implementaciju u računalu;
4. Izradu skice idejnog rješenja u obliku pseudokôda;
5. Implementaciju idejnog rješenja u obliku dijagrama toka i provjera korištenjem razvojne okoline Raptor [2];
6. Implementaciju i provjeru rješenja u programskom jeziku C♯ unutar razvojne okoline MS Visual Studio [3].

Kako najbolje koristiti ovu skriptu? Općenito postoje dva načina. Prvi je

uobičajeni način čitanja skripte od početka do kraja skripte, a drugi način je čitanje onih dijelova skripte koji su studentu trenutno potrebni za usvajanje specifičnog znanja. Zbog različite razine predznanja moguće je primijeniti oba načina čitanja knjige. Pri tome se studentima sa slabijim predznanjem iz područja računalstva preporučuje prvi način, a studentima s većim predznanjem drugi način korištenja ove skripte. Učenje programiranja traži usvajanja specifičnog načina razmišljanja u kojem je svaku vrijednost prije njenog korištenja potrebno spremiti u varijablu, svaka operacija se izvršava u obliku naredbe i računalo izvršava naredbu po naredbu točno kako ih je programer unio. Svakako se preporučuje svim studentima preuzimanje besplatne programske podrške dostupne na [2] i [3] te korištenje video snimki besplatnih predavanja dostupnih na [4]. Dostupne besplatne video snimke predavanja pokrivaju sva područja školovanja (matematika, kemija, umjetnost, programiranje i dr.) i uključuju forum kao platformu za brzu komunikaciju s kolegama iz cijelog svijeta koji usvajaju isto gradivo. Profesori predavači su s poznatih svjetskih sveučilišta, ali i sudjelovanje pojedinaca entuzijasta također je omogućeno.

1.1 Povijest razvoja računala

Današnja računala podržavaju obradu širokog spektra različitih podataka. Povijesni začetak razvoja računala potječe još iz doba antike kada su drevni narodi prvi puta osjetili potrebu za obradom podataka, njihovim spremanjem i mogućnosti kasnijeg ponovnog korištenja spremljenih podataka. Prva potreba za obradom podataka sastojala se od potrebe određivanja točnog datuma, odnosno godišnjeg doba i godine. Sve u svrhu održavanja vjerskih proslava i obreda u isto doba godine. Obrada podataka sastojala se od brojanja ili od gradnje posebnih građevina kod kojih su sunce i druge zvijezde svojim specifičnim položajem označavale određeni trenutak. Razvoj računala prema generacijama je prikazan u tablici 1.1.

Generacije računala dijele se prema tehnologijama koje su se koristile za ostvarivanje računanja, odnosno obrade podataka. Tehnologija nulte generacije zasnivala se na mehaničkim dijelovima (zupčanici, osovine, brojčanici) uz primjenu osnovnih električnih dijelova na kraju nulte generacije (releji i elektromotori za pogon računala). U ovoj generaciji već je postojala ideja da se napravi računalo s mogućnošću univerzalnog programiranja, no tehnološke mogućnosti toga vremena to nisu podržavale. Prva generacija računala omogućila je ostvarivanje računala s mogućnošću univerzalnog programiranja. Tehnologija koja se koristila bila je zasnovana na elektroničkim cijevima. Nedostatak je bila niska pouzdanost, velika potrošnja energije te velike dimenzije računala. Korisnici su bili vojska i državne službe. Druga generacija računala je kao tehnologiju

koristila tranzistor koji je donio smanjenje dimenzija i potrošnje energije te povećanje brzine rada i pouzdanosti. Također se pojavljuju i prva komercijalna računala koja su kupovala velike tvrtke te istraživačke institucije. Za komercijalna računala u velikim tvrtkama razvijeni su viši programski jezici COBOL i ALGOL, a za znanstvenu zajednicu FORTRAN. Tranzistori su se proizvodili pojedinačno na pločicama silicija. Povezivanjem tranzistora unutar iste pločice silicija nastaju integrirani krugovi, odnosno čipovi (engl. "chip") kao tehnologija treće generacije računala. Ovom tehnologijom je bilo moguće proizvesti cijele logičke sklopove na jednoj pločici silicija, što značajno smanjuje dimenzije računala kao i potrošnju energije. Računala postaju dovoljno moćna da poslužuju više od jednog korisnika pa se izrađuju terminali, odnosno jednostavno računalo sastavljeno od tipkovnice, monitora te mrežne kartice za vezu s glavnim poslužiteljem. Terminali su omogućavali korisniku da se spoji s glavnim računalom (poslužiteljem, engl. "server"), programira ga ili pokreće različite programe na njemu te da na svom zaslonu promatra rezultate izvršavanja pokrenutog programa. U ovoj generaciji računala se intenzivno koriste viši programski jezici kao što su COBOL, BASIC, ALGOL i FORTRAN. U to vrijeme nastaje viši programski jezik C kojeg odlikuje univerzalna primjena i naredbe bliske načinu rada mikroprocesora. Danas inačice programskog jezika C (C++ i C♯) čine najrasprostranjeniji viši programski jezik za izradu aplikacija za različite platforme (operacijske sustave).

Pojavom prvih mikroprocesora nastaje tehnologija četvrte generacije računala. Time jedan integrirani krug sadrži sve osnovne funkcionalnosti za univerzalnu obradu podataka, odnosno računanje kao što ga poznajemo iz današnjih računala. Pojavljuju se prva osobna računala i ona postaju dostupna za svakodnevno korištenje u malim tvrtkama i kućanstvima. Razvoj računalnih mreža i paralelne obrade podataka stvorio je temelj pete generacije računala. Pojava mikroprocesora s više jezgri i malom potrošnjom omogućila je stvaranje malih

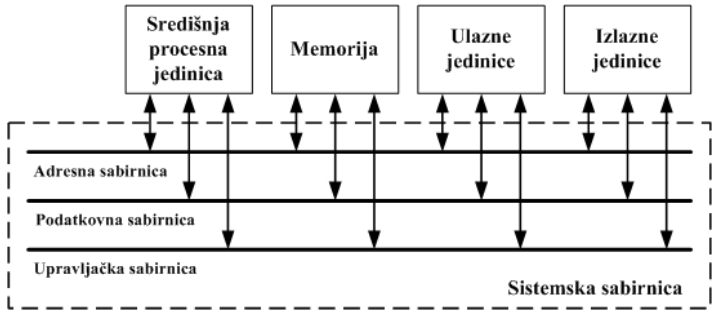
Generacija	Razdoblje	Tehnologija
Nulta	1642. - 1945.	Mehanički dijelovi (zupčanici i releji)
Prva	1945. - 1954.	Elektroničke cijevi
Druga	1954. - 1963.	Tranzistori
Treća	1963. - 1973.	Integrirani krugovi
Četvrta	1973. - 1985.	Integrirani krugovi vrlo visokog stupnja integracije
Peta	1985. - danas	Paralelna obrada i mreže
Šesta	U razvoju	Internet stvari i umjetna inteligencija

Tablica 1.1: Generacije razvoja računala.

prijenosnih računala s dugom autonomijom. Takvi mikroprocesori ugrađuju se u široku skupinu različitih uređaja tako da se pojavljuju inteligentna osjetila s lokalnom obradom podataka. Time nastaje osnova za izgradnju upravljačkih sustava za upravljanje velikim procesima kao što je to i prometni sustav. Trenutno je u razvoju sljedeća, šesta generacija računala. Naglasak je na sveopćem umrežavanju različitih uređaja te korištenju njihove procesorske snage za rješavanje složenijih problema na razini cijelog sustava. Za obradu podataka koriste se metode umjetne inteligencije koje računalu omogućuju autonoman rad u svojstvu vlastitog nadzora ispravnosti te automatske nadogradnje i rješavanja problema.

1.2 Građa računala

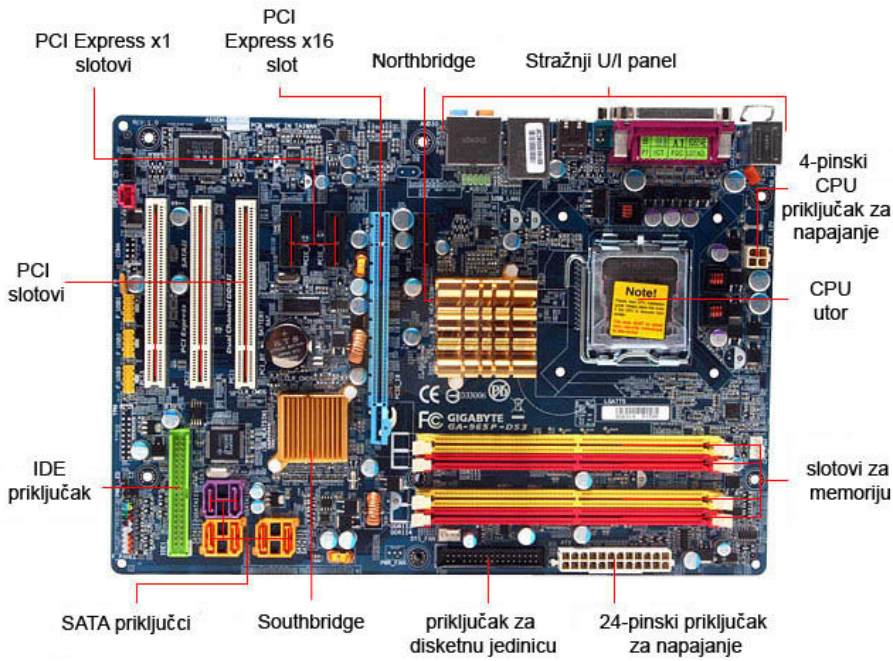
Kako bi obradilo podatke računalo prihvaća podatke, pretvara ih u odgovarajući prikaz te prikazuje rezultate obrade operateru u pogodnom obliku. Obrada podataka može se raditi u slijednom načinu rada (izvršava se naredba po naredba) ili u paralelnom načinu rada (više naredbi se izvršava istovremeno). Za obradu podataka potrebne su različite komponente računala pri čemu se osnovna podjela može napraviti na sklopovlje (engl. "hardware") i programsku podršku (engl. "software").



Slika 1.1: Von Neumann-ova arhitektura računala.

Sklopovlje računala se sastoji od različitih, međusobno povezanih jedinica [5]. John von Neumann u svom izvješću [6] opisuje računalo koje će svoj program imati zapisan u memoriji. Tim izvješćem opisana je arhitektura računala danas poznata kao Von Neumann-ova arhitektura, a blokovski je prikazana na slici 1.1. Ona se sastoji od središnje procesne jedinice (engl. "Central Processing Unit" - CPU) koja je pomoću sistemske sabirnice povezana s memorijom računala te ulaznim i izlaznim jedinicama. Sve ove komponente zajednički tvore jednu cjelinu koja može po volji mnogo puta izvršavati program spremljen u memoriju.

Svo sklopovlje je u današnjim računalima povezano preko matične ploče kao



Slika 1.2: Matična ploča računala [7].

što je prikazano na slici 1.2. Matična ploča predstavlja glavnu tiskanu pločicu u računalu koja povezuje sve komponente Von Neumann-ove arhitekture [8]. Sastoji se od dijelova objašnjenih u nastavku [7].

Čipovi poveznice (engl. "chipset") su dva čipa koji čine jezgru matične ploče. Prvi i veći čip, smješten bliže procesoru, čini sjevernu poveznicu (engl. "Northbridge"). Ona povezuje tri najbrže jedinice u računalu (procesor, memoriju i grafičku karticu) te nadgleda sav podatkovni promet koji se odvija među njima. Drugi i manji čip čini južnu poveznicu (engl. "Southbridge") koja nadgleda sav podatkovni promet koji se odvija između ostalih ulaznih i izlaznih jedinica.

Čip za BIOS (engl. "Basic Input Output System") služi za spremanje programa i podataka za inicijalizaciju svih jedinica prilikom uključivanja računala. Također upravlja procesom dohvaćanja i učitavanja operacijskog sustava računala.

Utor za procesor predstavlja dio matične ploče koji služi za prihvat procesora. Današnji procesori obično su u obliku malog tankog kvadra veličine pet

puta pet centimetara. Zbog potrebe za hlađenjem procesori su najčešće skriveni ispod većeg hladnjaka opremljenog ventilatorom.

Utori za radnu memoriju predstavljaju dio matične ploče koji služi za prihvat radne memorije. Radna memorija obično se proizvodi u obliku tiskanih pločica s više memorijskih čipova na njima.

Utori za unutrašnje jedinice služe za prihvat tiskanih pločica (kartica) koje računalu omogućuju dodatne funkcionalnosti kao što su komunikacija s drugim umreženim računalima, razmjena informacija s tehničkim procesom kojim računalo upravlja, snimanje i reprodukciju zvuka, prikaz slike i dr. Dio navedenih jedinica se u nekim modelima matičnih ploča ugrađuje u samu matičnu ploču u obliku čipa. Najčešće se radi o jedinicama za prikaz grafike, mrežnu komunikaciju te obradu zvuka. Time se smanjuju dimenzije računala, no takve ugrađene jedinice obično su nešto sporije.

Priključci za vanjsku memoriju služe za spajanje jedinica koje služe za trajnu pohranu većih količina podataka kao što su to tvrdi diskovi, optički (CD i DVD) pogoni i sl. Koriste se serijski (USB, SATA) i paralelni protokoli prijenosa podataka (IDE, SCSI).

Priključci za vanjske jedinice služe za spajanje jedinica koje se nalaze izvan kućišta računala kao što su to tipkovnica i miš, palica za igru, pisač i dr.

Pomoćni sklopovi služe za napajanje računala, hlađenje i sl.

1.2.1 Sistemska sabirnica

Kako je navedeno, pojedini dijelovi računala spojeni su sistemskom sabirnicom. Sabirnice su vodovi koji povezuju veći broj sklopova unutar čipa, sklopova na tiskanoj pločici, unutar računala ili povezuju vanjske jedinice s računalom [9]. Sastoje se od skupine više vodova pri čemu su promjene razina signala na pojedinim vodovima sinkronizirane. Sistemska sabirnica općenito se može sastojati od tri dijela:

Adresna sabirnica predstavlja dio sabirnice koji služi za prijenos signala koji definiraju adresu sklopa ili uređaja kojemu je podatak namijenjen, ili adresu u memoriji kojoj se želi pristupiti.

Podatkovna sabirnica predstavlja dio sabirnice koji služi za prijenos podatka između pojedinih sklopova spojenih na sabirnicu.

Upravljačka sabirnica predstavlja dio sabirnice s upravljačkim signalima koji aktiviraju sklopove koji sudjeluju u razmjeni podataka te sinkroniziraju prijenos podataka preko sabirnice.

Ovakva arhitektura računala, odnosno opisana konstrukcija sabirnice predstavlja i problem uskog grla prijenosa podataka. Radi se o fenomenu "Von Neumann Bottleneck". Naime, podatkovna se sabirnica istovremeno koristi za prijenos dvije vrste podataka, prijenos naredbi i samog podatka koji će se obraditi. Dok se radi prijenos samog podatka za obradu, središnja procesna jedinica je prisiljena čekati. Proces čekanja središnje procesne jedinice na podatak se izvodi tako da središnja procesna jedinica izvršava naredbu koja ne radi ništa (engl. "no operation" - NOP). Kako bi se vrijeme čekanja smanjilo, uz središnju procesnu jedinicu se smješta brza priručna memorija (engl. "cache") koja sadrži naredbe i potrebne podatke za izvođenje sljedećih nekoliko naredbi.

1.2.2 Središnja procesna jedinica

Središnja procesna jedinica ili središnja jedinica za obradu podataka predstavlja mozak računala, odnosno mjesto gdje se odvija sama obrada podataka [10]. Danas se za nju najčešće koristi skraćeni naziv procesor. Svaki procesor ima drugačiju građu (arhitekturu) i svoje pripadne naredbe ili instrukcije na raspologanju za izradu programa. Prema veličini skupa naredbi može se napraviti podjela procesora na procesore s velikim skupom složenijih naredbi (engl. "Complex Instruction Set Computing" - CISC) te na procesore s malim skupom jednostavnih naredbi (engl. "Reduced Instruction Set Computing" - RISC). Današnji procesori koriste komponente jedne i druge arhitekture, odnosno složenije se naredbe mogu izvršavati kao skup jednostavnijih procesorskih naredbi. Pri tome se procesori proizvode s više jezgri, odnosno svaka jezgra kao zasebni procesor može izvršavati naredbu nezavisno od druge jezgre čime je omogućeno paralelno izvršavanje naredbi.

Svaki procesor se sastoji od skupa registara opće namjene kao priručne memorije za spremanje podataka za obradu, skupa namjenskih registara za spremanje upravljačkih podataka, aritmetičko-logičke jedinice koja vrši obradu nad podacima te upravljačke jedinice koja nadzire rad procesora. Sve komponente procesora su međusobno povezane pripadnom sabirnicom. Prilikom izvršavanja programa procesor dohvaća iz memorije naredbu po naredbu i izvršava ju. Izvođenje naredbi odvija se unutar ciklusa pri čemu općeniti ciklus izvođenja naredbi sadrži korake opisane u nastavku.

Dohvat naredbe iz memorije (engl. "fetch") predstavlja korak u kojem se dohvaća naredba iz memorije računala. Adresa s koje se dohvaća naredba je spremljena u namjenskom registru programsko brojilo (engl. "Program Counter"). Ovaj se registar nakon završetka ovog koraka povećava za vrijednost 1 kao priprema za dohvat sljedeće naredbe. Naredba se sprema u namjenski registar instrukcija. Ovaj korak je identičan za sve naredbe.

Analiza sadržaja naredbe (engl. "decode") predstavlja korak u kojem se dohvaćena naredba prevodi i priprema za izvođenje. Za prevođenje naredbe koristi se sklop dekoder pri čemu se prepoznaje vrsta naredbe. U slučaju da se izvodi naredba grananja ili naredba koja zahtjeva dodatne podatke (npr. zbrajanje zahtjeva najmanje dva pribrojnika), prilagođava se vrijednost spremljena u registru programsko brojilo.

Izvršavanje naredbe (engl. "execute") predstavlja korak u kojem se naredba izvodi u dijelovima procesora aktiviranim pomoću upravljačke jedinice. Obrada podataka se odvija u aritmetičko-logičkoj jedinici. Rezultat obrade se sprema u registar opće namjene koji se naziva akumulator.

Brzina izvođenja naredbi zavisi od takta procesora, odnosno od frekvencije na kojoj radi procesor. Pri tome se na navedenoj frekvenciji procesora izvršavaju pojedini koraci ciklusa naredbi. Posljedica je da se naredbe izvršavaju na frekvenciji oko 4 puta manjoj od frekvencije na kojoj radi procesor. Potrebno je imati na umu da je kod nekih naredbi u koraku izvršavanje naredbe potrebno pristupiti memoriji radi dohvata dodatnih podataka za obradu što zahtjeva jedan dodatni korak. Kako brzina izvršavanja cijele naredbe ne bi bila oko 4 puta manja od takta procesora, paralelno se izvršavaju pojedini ciklusi naredbi u sklopu tzv. cjevovoda izvršavanja (engl. "instruction pipelane"). Dok se odvija korak izvršavanja trenutne naredbe, sljedeća naredba se prevodi i odvija se dohvat naredbe koja će se izvesti nakon sljedeće naredbe. U slučaju izvođenja slijednih programa, odnosno uzastopnih naredbi, tako se dobiva značajno ubrzanje. Problem nastaje u slučaju grananja programa kada se preskače dio naredbi što znači da je iz cjevovoda izvršavanja potrebno odbaciti dio naredbi pripremljenih za izvršavanje.

1.2.3 Memorija

Memorija računala služi za spremanje svih vrsta podataka u računalu. Za spremanje podataka u računalu koriste se dva različita stanja, odnosno dvije znamenke binarnog brojevnog sustava. Stanja su označena s 0 i 1, a jedno stanje je moguće spremiti u jedan bit (skraćeno od engl. "Binary Digit"). Radi lakšeg spremanja podataka, radi se grupiranje bitova pri čemu skupina od n bitova može prikazati 2^n različitih podataka. Bitovi se najčešće grupiraju u skupine od 8 bitova, odnosno u bajtove ili oktete (engl. "byte"). Svaki bajt u memoriji ima svoju jedinstvenu adresu i predstavlja najmanju lokaciju kojoj se može adresirati pristup. Za prikaz podataka često je potrebno iskoristiti više od jednog bajta, pri čemu onda adresa prvog bajta predstavlja adresu podatka.

Bajt se također koristi kao jedinica za mjerenje veličine memorije. Za bajt

se koristi oznaka B, a za bit b. Radi lakšeg prikaza veće količine memorije koriste se prefiksi uobičajeni za prikaz fizikalnih veličina. Problem je što se za prikaz fizikalnih veličina koristi brojevni sustav s bazom 10, a računala koriste brojevni sustav s bazom 2. Količina koju uobičajeni prefiksi označavaju se malo razlikuje kada se one primijene u računalstvu. Iznos razlike moguće je vidjeti u tablici 1.2. Razlika u količini memorije postaje sve značajnija s povećanjem količine memorije.

Prikaz fizikalnih veličina			Prikaz količine memorije		
Prefiks	Oznaka	Količina	Prefiks	Oznaka	Količina
kilo	k	10^3	kibi	Ki	$2^{10} = 1.024$
mega	M	10^6	mebi	Mi	$2^{20} = 1.048.576$
giga	G	10^9	gibi	Gi	$2^{30} = 1.073.741.824$
tera	T	10^{12}	tebi	Ti	$2^{40} = 1.099.511.627.776$

Tablica 1.2: Prikaz količine memorije.

Prema vrsti memorije može se napraviti sljedeća podjela [10, 7]:

Ispisna memorija (engl. "Read Only Memory" - ROM) predstavlja memoriju u koju se podaci spremaju samo jednom prilikom njene proizvodnje. Korisnik nema mogućnosti zapisivanja podataka u ovu vrstu memorije. U ovu vrstu memorije spremaju se bitni podaci koji ostaju nepromijenjeni tijekom cijelog radnog vijeka računala, kao što je to npr. BIOS računala. On postavlja osnovne radne parametre računalnog sklopovlja te pronalazi i učitava operacijski sustav u radnu memoriju. Razvojem računala pojavila se potreba za dodatnim vrstama ROM memorije pa su tako nastale ispisne memorije u koju korisnik može jednom zapisati podatke (engl. "Programable Read Only Memory" - PROM), ispisne memorije s mogućnošću brisanja pomoću sunčeve svjetlosti (engl. "Erasable PROM" - EPROM) ili one kod kojih se brisanje postiže dovođenjem povećanog napona na priključke memorije (engl. "Electricaly Erasable PROM" - EEPROM). Nakon brisanje se u memoriju može zapisati novi podatak.

Memorija sa slučajnim pristupom (engl. "Random Access Memory" - RAM) naziva se i radna memorija. Služi u računalu kao glavna memorija za privremenu pohranu podataka pri čemu sadržaj memorije ostaje sačuvan samo dok postoji napajanje. Odlikuje ju malo vrijeme pristupa (engl. "access time") podacima, neovisno u kojem dijelu memorije su podaci spremljeni. Koristi se izravan pristup podacima spremljenim u memoriji bez potrebe za pozicioniranjem glave za čitanje, odnosno pisanje. Izrađuje se od poluvodičkih sklopova, a obično se teži da računalo sadrži što veću količinu radne memorije.

Brza priručna memorija (engl. "Cache") je memorija brža od klasične radne memorije i služi za pohranjivanje malih dijelova radne memorije, odnosno programa i podataka prije njihove neposredne obrade. Koriste se za izjednačavanje različite brzine rada CPU-a i ostalih jedinica računala. Danas se ona ugrađuje u sve jedinice računala tako da se podaci prije prijenosa sabirnicom spajaju u blok podataka koji se sprema u brzu priručnu memoriju.

Magnetska memorija je memorija za trajnu pohranu podataka pri čemu se kao medij za pohranu podataka koriste magnetske vrpce i diskovi. Omogućuju spremanje velike količine podataka u slijednom načinu rada. To znači da se podaci zapisuju u nizu tako da je za pristup podacima potrebno prvo pozicionirati glavu za čitanje, odnosno pisanje. Nekada su se koristile magnetske vrpce, a danas diskovi. Najpoznatiji predstavnik je tvrdi disk (engl. "hard disc") kojeg računala koriste kao glavnu memoriju za trajnu pohranu podataka.

Optička memorija (engl. "Optical memory") za trajnu pohranu i čitanje veće količine podataka koristi svjetlosna svojstva materijala. Pohrana i čitanje podataka izvode se pomoću lasera koji ima sposobnost očitanja i zapisa podataka na optički disk. Prva inačica ove memorije bio je CD (engl. "Compact Disc") na koji su se tvornički zapisivali podaci pa u procesu proizvodnje nastaje tzv. CD ROM (engl. "CD Read Only Memory"). Razvojem nastale su inačice na koje je korisnik mogao jednom (engl. "CD Recordable" - CD-R) ili više puta (engl. "CD Rewritable" - CD-RW) zapisati podatak te inačice s većim kapacitetom memorije (engl. "Digital Versatile Disc" ili "Digital Video Disc" - DVD i engl. "Blu-Ray Disc").

Flash memorija (engl. "Flash memory") je vrsta poluvodičke memorije koja omogućuje trajnu pohranu podataka. Njihova osnovna karakteristika je da je za zapis novih podatka ili promjenu postojećih podataka prvo potrebno izbrisati cijelo memorijsko područje u koje će se spremiti novi podaci. Time se značajno usporava operacija spremanja podataka u ovu vrstu memorije. Danas se ona koristi za proizvodnju lako prenosivih medija za pohranu podataka kao što su to memorijske kartice, USB (engl. "Universal Serial Bus") memorije i diskovi bez pokretnih dijelova (engl. "Solid-State Drive" - SSD).

Prividna ili virtualna memorija (engl. "Virtual Memory") je memorija koja se koristi za proširenje radne memorije. Smješta se u sklopu neke druge vrste memorije, a najčešće na tvrdi disk računala. Ideja prividne memorije jest omogućiti računalu da raspolaže većom količinom radne memorije, nego što je u računalo ugrađeno i koju može po potrebi povećavati ili

smanjivati. Nedostatak je što je ovaj dio memorije višestruko sporiji od radne memorije.

1.2.4 Ulazne i izlazne jedinice računala

Računalo pomoću različitih jedinica razmjenjuje podatke sa svojom okolinom, odnosno ulazne i izlazne jedinice računala omogućuju korištenje računala i interakciju s korisnikom. Okolinu računala čine korisnik ili operater, druga računala povezana nekom komunikacijskom vezom, tehnički proces kojim računalo upravlja i sl. Podjela jedinica se radi prema smjeru protoka podataka kao što je navedeno u nastavku.

Ulazne jedinice karakterizira smjer podataka od okoline prema računalu. One omogućuju unos podataka u računalo. Kao primjeri se mogu navesti tipkovnica, miš, optički pogoni samo s mogućnošću čitanja (CD-ROM i DVD-ROM), digitalni fotoaparat, analogno/digitalni (A/D) pretvornik (pretvara pripadnu analognu električnu veličinu izmjerene vrijednosti fizikalne veličine u digitalnu veličinu (numeričku informaciju) pogodnu za obradu u računalu) i sl.

Izlazne jedinice karakterizira smjer podataka od računala prema okolini. One omogućuju prikaz podataka koji se nalaze u računalu, odnosno prikaz rezultata obrade unesenih podataka. Kao primjeri mogu se navesti zaslon (monitor), pisač (engl. "printer"), crtalo (engl. "plotter"), video projektor, digitalno/analogni (D/A) pretvornik (pretvara numeričku vrijednost u računalu u analognu vrijednost pogodnu za prijenos do izvršnog člana pomoću kojeg se djeluje na neki upravljani proces) i sl.

Ulazno-izlazne jedinice karakterizira dvosmjerna razmjena podataka. Istovremeno su podržana oba smjera protoka podatka, od okoline u računalo i obrnuto. Kao primjeri mogu se navesti zvučna (audio) kartica (na nju se spaja mikrofon kao ulazna informacija i zvučnik kao izlazna informacija), zaslon osjetljiv na dodir (prikaz podataka u računalu i upravljanje računalom), optički pogoni koji mogu čitati i pisati podatke, mrežne kartice i sl.

1.3 Programska podrška

Kao što je već navedeno za obradu podataka pomoću računala koriste se sklopovlje i programska podrška kao cjelina. Programska podrška sastoji se od niza naredbi spremljenih u trajnoj memoriji, najčešće tvrdom disku. Tijekom

obrade podataka se prvo u radnu memoriju računala učita programska podrška, a nakon toga se dohvaćaju podaci za obradu te rezultati obrade spremaju u memoriju računala. Programska podrška se može podijeliti na operacijske sustave i korisničke aplikacije.

1.3.1 Operacijski sustavi

Operacijski sustav računala predstavlja programsku podršku koja upravlja i nadgleda sve operacije računala [8]. Omogućuje korisniku da stvara, kopira i briše datoteke, da zadaje naredbe računalu korištenjem tekstualnog ili grafičkog sučelja te stvara okruženje koje omogućuje nesmetanu obradu podataka. U tom cijelom procesu korištenja računala korisnik ne vodi računa o tome kako inicijalizirati neku jedinicu računala kako bi se ona mogla koristiti, brzini prijenosa podataka između pojedinih jedinica kako bi cijeli sustav bio usklađen, instalaciji potrebnih programskih nadogradnji, raspodjeli procesa obrade između pojedinih jezgri višejezgrenog procesora i sl. Sve te zadatke izvršava operacijski sustav bez potrebe za djelovanjem od strane korisnika.

Razvoj operacijskih sustava bio je potaknut potrebom za pojednostavljenjem korištenja računala. Naime, na početku razvoja računala svaki je korisnik morao zasebno unijeti svoj program te pokrenuti računalo da se on izvrši. To je zahtjevalo veliko znanje budući da je bilo potrebno u programu napisati programski kôd na najnižoj strojnoj razini, odnosno svaka naredba se unosila pisanjem pripadnog binarnog kôda. Ta razina je uključivala podešavanje svake jedinice računala u sklopu programskog kôda. Računalo je isto tako mogao koristiti samo jedan korisnik bez mogućnosti utjecaja na izvršavanje programa. Uvođenje operacijskog sustava omogućilo je korisniku da se koncentrira na samo korištenje računala kao alata za obradu podataka, a ne na njegovo programiranje. Današnji operacijski sustavi imaju grafičko korisničko sučelje te omogućuju istovremeno izvršavanje više korisničkih aplikacija. Poslužitelji omogućuju i istovremeni rad više korisnika pri čemu se korisnik spaja na poslužitelj pomoću terminala ili odgovarajuće korisničke aplikacije. Kao primjeri operacijskih sustava mogu se navesti MS Windows, Android, Mac OS, UNIX, Solaris, Linux i dr.

1.3.2 Korisničke aplikacije

Obrada podataka pomoću računala obavlja se korištenjem odgovarajuće programske podrške, odnosno korisničke aplikacije koju izvodi operacijski sustav. Pri tome operacijski sustav predstavlja jezgru koja nadgleda izvođenje korisničke aplikacije te upravlja sklopovljem računala na način da se korisnička aplikacija može nesmetano izvoditi. Tako operacijski sustav olakšava korištenje računala,

a korisničke aplikacije olakšavaju izvršavanje određenog radnog zadatka na računalu. Korisničke se aplikacije rade za određeni radni zadatak pa tako postoje aplikacije za obradu dokumenata (npr. MS Office, Open Office, WPS Office), izradu tehničkih nacrta (npr. MS Visio, AutoCAD, VectorEngineer), izradu simulacijskih modela te simuliranje različitih tehničkih i tehnoloških procesa (npr. Matlab, VISSIM, VISUM, AimSun, MathSim, SUMO, ARIS), baze podataka (npr. MS Access, MySQL, MS SQL, Oracle, Firebird, IBM Informix), razvojne okoline za izradu programske podrške (npr. MS Visual Studio, Raptor, Eclipse, KDevelop), praćenje i vođenje projekata (npr. MS Project, ProjectLibre, 2-plan Project Management Software) itd.

Svaka korisnička aplikacija ima svoj vlastiti format zapisa podataka, odnosno rezultirajućih dokumenata ili projekata. Radi povećanja kompatibilnosti koriste se definirani formati zapisa koje aplikacije više različitih proizvođača mogu koristiti i obrađivati. Na primjer, kod obrade slika definirani su formati *jpg*, *bmp*, *png* i dr. koji svaka korisnička aplikacije za obradu slike može učitati. Također postoje aplikacije koje rade pretvorbu između različitih formata zapisa. Na primjer kod obrade teksta često se koristi pretvorba između formata *docx* i *pdf*.

Nekada su se aplikacije kupovale i instalirale na korisničko računalo. Mogle su se koristiti samo na računalu na kojem su bile instalirane. Pojavom tehnologije oblaka (engl. "cloud") to se značajno promijenilo, tako da je danas moguće koristiti aplikacije koje se izvode na udaljenom računalu, a korisnik ne primjećuje razliku u grafičkom sučelju. Udaljeno računalo ima memorijski prostor rezerviran za korisnika te aplikacije. Memorijski prostor zakupljuje korisnik prema vlastitim potrebama. Programi i podaci koje korisnik obrađuje spremljeni su na udaljenom računalu i dostupni s bilo kojeg mjesta u svijetu uz uvjet da postoji pristup Internetu i računalo s nekim web preglednikom (engl. "web browser) kao što su MS Internet Explorer, Mozilla Firefox, Opera, Chrome i dr.. Ova tehnologija također omogućuje da više korisnika u isto vrijeme radi na istom dokumentu. Sve njihove promjene sadržaja dokumenata automatski se sinkroniziraju i postaju vidljive svim korisnicima s odgovarajućim ovlastima.

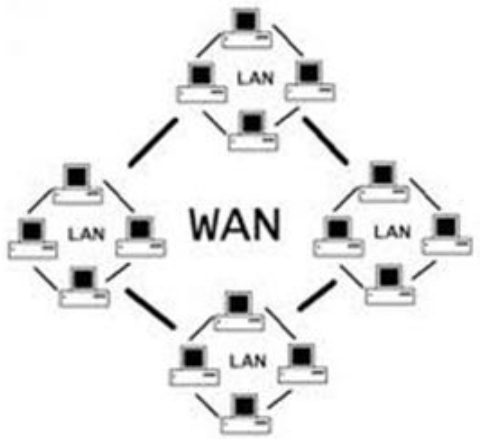
1.4 Računalne mreže

Prva su računala bile zasebne jedinice. Operater je u njih unio program te podatke za obradu i pričekao rezultat obrade. U današnjem svijetu računala su umrežena tako da mogu izmjenjivati podatke između sebe, odnosno omogućuju korisniku da se spoji na udaljeno računalo te na njemu radi obradu svojih podataka. Bitno je samo opremiti računalo odgovarajućom mrežnom karticom te instalirati program s pripadnim protokolom za komuniciranje.

Primjena računalnih mreža naročito je bitna u upravljanju velikim i složenim tehničkim sustavima, kao što je to prometni sustav. Današnji tehnički sustavi su opremljeni većom količinom osjetila i upravljačkih uređaja. Pri tome se koriste pametna osjetila (sastavljena od klasičnog osjetila i malog računala s mrežnom karticom za osnovnu obradu podataka) te upravljački uređaji s mogućnošću komuniciranja što stvara veliku mrežu rasprostranjenih malih računala. Dodatno, danas skoro svaki korisnik mobilne komunikacijske mreže koristi pametni telefon koji također ima u sebi ugrađeno osjetilo za mjerenje položaja te brzine kretanja tako da svaki korisnik može dati upravljačkom uređaju korisne mjerne podatke o trenutnom stanju prometne mreže u njegovoj okolini.

1.4.1 Vrste računalnih mreža

Računalne mreže mogu se podijeliti na različite načine [11]. Prva podjela računalnih mreža je prema veličini pa se razlikuju lokalne računalne mreže (engl. "Local Area Network" - LAN) te mreža širokog (globalnog) područja (engl. "Wide Area Network" - WAN). Grafički prikaz odnosa između LAN i WAN računalne mreže je dan na slici 1.3.



Slika 1.3: Odnos između lokalne i globalne računalne mreže.

Lokalnu računalnu mrežu čine dva ili više računala povezana prijenosnim medijem unutar zgrade ili ustanove. Vezane su za malo lokalno područje ureda, odnosno područje smještaja ustanove ili tvrtke. Lokalnu mrežu moguće je podijeliti na Intranet te Ekstranet računalnu mrežu. Intranet se još naziva i lokalnom Internet mrežom što znači da pristup njoj imaju samo autorizirani korisnici. Obično su to zaposlenici tvrtke ili članovi neke zajednice. Na taj se način štiti sigurnost interne dokumentacije i pripadnih podataka. Kao primjer može se na-

vesti portal FPZnet kojem pristup imaju samo zaposlenici Fakulteta prometnih znanosti Sveučilišta u Zagrebu. Ekstranet računalna mreža omogućuje pristup Intranetu i biranim vanjskim korisnicima. Obično su to korisnici usluga neke institucije ili tvrtke. Kao primjer može se navesti e-Student kojem pristup imaju zaposlenici i studenti Fakulteta prometnih znanosti Sveučilišta u Zagrebu. Pri tome zaposlenici imaju veća prava pristupa od studenata. Ostali korisnici nemaju dozvolu pristupa.

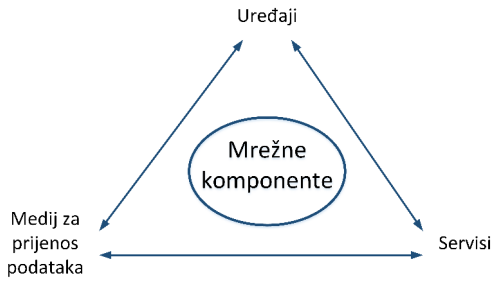
Mrežu širokog (globalnog) područja odnosno WAN obično čini nekoliko međusobno povezanih lokalnih računalnih mreža. Obuhvaća veće zemljopisno područje kao što je to grad ili cijela država. Najpoznatija izvedba globalne računalne mreže je Internet i na tu mrežu je danas povezana većina računala.

Druga podjela računalnih mreža je prema prijenosnom mediju koji se koristi za prijenos podataka. Mogu se podijeliti na žičane računalne mreže te bežične računalne mreže. Žičane računalne mreže za prijenos podataka koriste kablove izrađene od bakrene žice ili optičkih vlakana. Pri tome optička vlakna omogućuju postizanja većih brzina prijenosa podataka i otpornija su na smetnje pa se optički komunikacijski kabel može položiti u istu kanalicu u koju je položen energetski kabel. Bežične računalne mreže za prijenos podataka koriste elektromagnetske valove. Bežični ekvivalent lokalnoj računalnoj mreži je tzv. bežična lokalna računalna mreža (engl. "Wireless LAN" - WLAN). Najraširenija inačica je Wi-Fi i danas podržava brzine prijenosa do 7 Gb/s. U prometu je još bitna i bežična mreža kratkog dometa (do otprilike 10 m) Bluetooth. Podržava brzine prijenosa do 1 Mb/s te omogućuje jednostavno i brzo povezivanje različitih vrsta uređaja. Kako je vrlo česta (nalazi se u svakom mobilnom komunikacijskom uređaju i prijenosnom računalu), omogućuje praćenje korisnika, odnosno vozila unutar prometne mreže. Praćenjem dovoljnog broja korisnika moguće je napraviti i dovoljno dobru estimaciju cestovnih prometnih parametara kao što su brzina, tok, izvorišno-odredišne matrice i dr.

1.4.2 Mrežne komponente

Kako bi se izgradila računalna mreža, potrebne su tri mrežne komponente: (i) uređaji; (ii) medij za prijenos podataka; i (iii) mrežni servisi (usluge). Pri tome se uređaji i medij za prijenos podataka odnose na mrežnu opremu. Odnos između navedenih mrežnih komponenata prikazan je na slici 1.4.

Uređaji se mogu podijeliti na krajnje uređaje (engl. "network edge") i uređaje posrednike u komunikaciji (engl. "network core"). Krajnji uređaji predstavljaju ishodište ili odredište podataka koji se šalju računalnom mrežom. Pri tome isti krajnji uređaj može preuzeti ulogu korisnika (engl. "client", računalo koje pristupa servisima (uslugama) na drugom udaljenom računalu i potražuje



Slika 1.4: Odnos između mrežnih komponenata.

podatke), poslužitelja (engl. "server", računalo s instaliranom programskom podrškom koje pruža uslugu ili podatke korisniku) ili obje uloge. Model rada Interneta je zasnovan na strukturi korisnik/poslužitelj odnosno na korištenju distribuiranih aplikacija i podataka. Tako je svaka web stranica smještena na poslužitelju koji je za korisnika udaljeno računalo. Prilikom svakog pristupa web stranici mi smo korisnici koji pristupaju udaljenom računalu.

Uređaji posrednici čine jezgru računalne mreže i omogućuju dvosmjernu razmjenu podataka, a moguće ih je podijeliti na sljedeće uređaje:

Mrežna kartica (engl. "Network Interface Card" - NIC) koja se ugrađuje u računalo. Pretvara podatke u računalu u odgovarajući električni, svjetlosni ili radijski signal ovisno o prijenosnom mediju radi prijenosa podataka. Svaka mrežna kartica ima svoju jedinstvenu fizičku (engl. "Media Access Control" - MAC) adresu koja se sastoji od 48 bitova.

Usmjernik (engl. "router") omogućava komunikaciju između računalnih mreža te njihovo segmentiranje. Izvodi se kao zaseban uređaj te pronalazi najbolji put za prijenos podataka između danog ishodišta i odredišta.

Preklopnik (engl. "switch") mikrosegmentira računalnu mrežu da svim računalima omogući korištenje pune brzine računalne mreže. Također se izvodi kao zaseban uređaj, a svako se umreženo računalo priključuje na zaseban utor (engl. "port").

Prevodilac protokola (engl. "gateway") se nalazi u čvoru računalne mreže i služi za povezivanje računalnih mreža koje koriste različiti protokol komunikacije. Bitan je za povezivanje računala s Ethernet mrežnom karticom s nekom drugom računalnom mrežom kao što je to npr. CAN (engl. "Controller Area Network") mreža koja je vrlo česta u vozilima.

Medij za prijenos podataka služi za prijenos, odnosno odašiljanje i primanje električnih te elektromagnetskih signala koji služe za prijenos podataka. Pri tome signali mogu biti analogni ili digitalni, a sama informacija je kôdirana. Kod analognih signala kôdiranje se radi promjenom (modulacijom) amplitude, frekvencije ili faze signala, a kod digitalnih signala se koristi zapis u binarnom kôdu.

Mrežni servisi (usluge) služe kao potpora korištenju različitih aplikacija na Internetu. Kao primjer mrežnih servisa (usluga) zasnovanih na razmjeni poruka između računala korisnika i poslužitelja se može navesti:

HTTP (engl. "Hypertext Transfer Protocol") utvrđuje način prijenosa dokumenata između poslužitelja i preglednika za prikaz web stranica. Većini web stranica pristupa se pomoću ovog protokola.

DNS (engl. "Domain Name System") predstavlja distribuirani hijerarhijski sustav Internet poslužitelja u kojem se nalaze informacije povezane s domenskim nazivima odnosno jedinstvenim adresama pojedinih računala. Definira povezanosti IP (engl. "Internet Protocol") adresa računala i pripadnih simboličkih imena računala.

DHCP (engl. "Dynamic Host Configuration Protocol") služi za automatsko dodjeljivanje IP adresa i ostalih mrežnih postavki računalu koje se prijavilo na mrežu. Olakšava spajanje računala na mrežu kako nije potrebno ručno podesiti mrežne postavke za svaku mrežu, već se računalo automatski podešava za pojedinu mrežu.

FTP (engl. "File Transfer Protocol") predstavlja protokol i program koji služi kao korisničko sučelje za prijenos datoteka. Postoje privatni i javno dostupni FTP poslužitelji. Kod javnih FTP poslužitelja često se kao korisničko ime koristi "anonymous", a kao zaporka vlastita adresa elektroničke pošte. Danas se FTP protokol rijetko koristi samostalno, već je ugrađen u suvremene web preglednike tako da se razmjena podataka korištenjem FTP protokola odvija nevidljivo za korisnika.

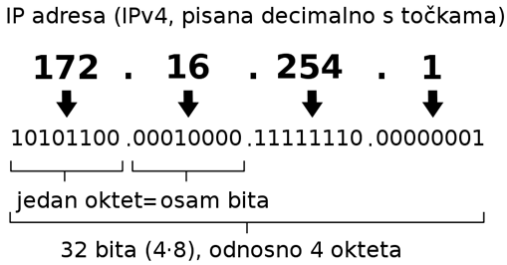
Elektronička pošta (engl. "e-mail") predstavlja sustav za brzu i jednostavnu razmjenu poruka i multimedijalnih dokumenata. Svakom korisniku se dodjeljuje vlastita jednoznačna adresa u obliku "ime.prezime@racunalo.domena" pri čemu znak "@" znači da se radi o korisniku na određenom računalu. Za prijenos elektroničke pošte koriste se protokoli SMTP (engl. "Simple Mail Transfer Protocol"), POP (engl. "Post Office Protocol"), IMAP (engl. "Internet Message Access Protocol") te HTTP protokol za pristup preko web sučelja.

1.4.3 Ethernet

Većina današnjih lokalnih računalnih mreža se uspostavlja korištenjem Ethernet mreže ili nekom njenom inačicom. Norma za Ethernet mrežu je objavljena 1980. godine od strane konzorcija tvrtki DEC, IBM i Digital. Značajke ove računalne mreže su:

- Jednostavnost i lakoća korištenja;
- Mogućnost povezivanja novih tehnologija;
- Pouzdanost;
- Niska cijena instalacije i održavanja;
- Slučajan pristup računala mediju za prijenos podataka;
- Proširenje na stvarnovremensku komunikaciju za industrijske primjene.

Ethernet mreža danas se najčešće koristi za pristup Internetu. Pri tome svako računalo prilikom prijave dobiva svoju jedinstvenu logičku IP adresu. Unutar DNS poslužitelja svakoj se numeričkoj IP adresi pridružuje simbolička adresa radi lakšeg razlikovanja računala od strane vanjskog korisnika (čovjeka). Za kreiranje IP adresa danas su aktualne norme IPv4 i IPv6. U normi IPv4 IP adresa definirana je kao binarni broj dug 32 bita podijeljen na četiri 8-bitna broja raspona $[0, 255]$. Radi lakšeg pamćenja IP adresa se često zapisuje u dekadskoj notaciji, odnosno kao četiri troznamenkasta dekadska broja odvojena točkama. Primjer zapisa jedne IP adrese po normi IPv4 prikazan je slikom 1.5. Prvi dio bitova IP adrese određuje klasu računalne mreže, a ostali bitovi određuju računala na računalnoj mreži. Klase računalne mreže se označuju slovima A do E.



Slika 1.5: Primjer IPv4 adrese.

Prema normi IPv6 IP adresa definirana je kao binarni broj dug 128 bitova. Ovo proširenje je uvedeno zbog naglog povećanja broja korisnika računalnih mreža. Naime, svaki pametni telefon, inteligentno osjetilo, ugradbeno računalo, upravljačko računalo itd. danas je umreženo i potrebno mu je dodijeliti IP

adresu. Uz ovo proširenje postoji mogućnost dodjele $5 \cdot 10^{28}$ različitih adresa. Poboljšanja IPv6 norme prema normi IPv4 uključuju:

- Mnogo veći adresni prostor;
- Novi format zaglavlja;
- Ugrađen mehanizam zaštite podataka;
- Poboljšanu podršku za kvalitetu usluge;
- Proširivost.

IP adresa prema normi IPv6 zapisuje se u heksadecimalnom brojevnom sustavu kao četiri znamenke međusobno odvojene dvotočkom, npr. $2001 : 0DB8 : 0000 : 0000 : 0000 : 0000 : 1428 : 57ab$. Zbog skraćenog zapisa IPv6 adrese vodeće nule unutar skupine od četiri znamenke mogu se ispustiti, odnosno ako se o radi skupini od četiri nule moguće ih je potpuno ispustiti pa se dobiju dvije dvotočke zaredom. Po tom pravilu navedeni se primjer IPv6 adrese može prikazati kao $2001 : DB8 :: 1428 : 57ab$. Radi očuvanja kompatibilnosti danas računala imaju paralelno IP adrese u normi IPv4 i u normi IPv6.

Poglavlje 2

Prikaz i pretvorba brojeva

Računalo kao alat predstavlja univerzalan stroj za obradu podataka prikazanih u numeričkom obliku. Pri tome se svaki podatak, koji u svom prirodnom prikazu nije numerički (npr. tekst, vrste boja, oblici objekata koji nas okružuju, vrste automobila i dr.), preslikava u odgovarajući numerički format odnosno kôdira. Navedeno preslikavanje odnosno pretvorba u skupinu brojeva odvija se prema odgovarajućim pravilima definiranim za svaki tip podatka posebno. Kako su se za brojanje prvo koristili ljudski prsti na rukama kojih ukupno ima 10, prvi korišteni brojevni sustav je bio onaj s bazom 10, odnosno dekadski. Iz istog razloga se u komunikaciji među ljudima zadržao do danas, dok strojevi koriste brojevne sustave s drugačijim bazama. Iz tehnoloških je razloga računalu najlakše razlikovati dva različita stanja (ima napona ili nema napona) pa koriste binarni brojevni sustav [9]. Binarni brojevni sustav zahtjeva stoga veći broj znamenaka za prikaz broja u usporedbi s dekadskim pa je nepregledan. Stoga se za prikaz podataka operateru koriste brojevni sustavi s većim bazama.

2.1 Brojevni sustavi

Za prikaz numeričkih podataka odnosno brojeva potrebno je definirati odgovarajući brojevni sustav, odnosno definirati način zapisivanja brojeva i njihovo tumačenje [5]. Danas se koristi tzv. položajni (pozicijski) brojevni sustav. Za tumačenje broja prikazanog u položajnom brojevnom sustavu bitan je položaj pojedine znamenke a_i unutar broja. Položaj znamenke definira njenu težinu koja se određuje bazom b brojevnog sustava. Pri tome baza brojevnog sustava definira i skup znamenki $\boldsymbol{Z}$ koji se može koristiti za prikaz brojeva. Ukupan broj

različitih znamenki N_z jednak je bazi b, a skup znamenki počinje od vrijednosti 0 pa do vrijednosti $b-1$. To se može zapisati na sljedeći način:

$$\boldsymbol{Z} = \{z_1, z_2, ..., z_{N_z}\} = \{0, 1, ..., b-1\}. \tag{2.1}$$

Numerička informacija najčešće ne sadrži samo cijeli dio broja pa se koristi decimalni zarez kako bi se odvojio cijeli (dekadski) i decimalni dio broja. Bitno je ovdje naglasiti da se kod prikaza brojeva u programskim jezicima koristi decimalna točka dok se u hrvatskom pravopisu koristi decimalni zarez. Koristeći navedenu notaciju općeniti numerički podatak P u bazi b se može prikazati kao [5]:

$$P = (a_2 a_1 a_0, a_{-1} a_{-2})_b = a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2}. \tag{2.2}$$

Promotri li se izraz 2.2 moguće je primijetiti da lijeva znamenka ima najveću težinu, da znamenka neposredno s lijeve strane decimalnog zareza ima težinu 1 ($b^0 = 1$ za bilo koju vrijednost baze) te da težine s desne strane decimalnog zareza imaju negativnu potenciju za prikaz decimalnog dijela numeričkog podatka. Iz tog razloga lijevu znamenku nazivamo najznačajnija znamenka, a desnu znamenku nazivamo najmanje značajnom znamenkom. U prikazu broja navedena je baza u kojoj je prikazan broj kako bi se u računalstvu izbjegla zabuna budući da se koristi nekoliko različitih baza. U slučaju da baza nije označena pretpostavlja se da je broj naveden korištenjem dekadskog brojevnog sustava, odnosno koristi se baza brojevnog sustava 10.

Raspon brojeva koji je moguće prikazati određenim brojem znamenki ovisi o bazi brojevnog sustava te o broju znamenki. Pri tome je pretpostavka da svaka znamenka doprinosi vrijednosti prikazanog broja prema navedenoj definiciji položajnog brojevnog sustava. Na taj je način moguće prikazati samo pozitivne brojeve. To je moguće prikazati sljedećim izrazom:

$$R = [0, b^n - 1], \tag{2.3}$$

gdje n označava broj znamenki korišten za prikaz broja. U prikazu numeričkih podataka često se postavlja pitanje koja najmanja količina znamenaka može prikazati određeni podatak, odnosno traženi raspon vrijednosti. Tu informaciju najlakše je odrediti tako da se broj koji se želi prikazati u nekoj drugoj bazi logaritmira u toj bazi. Dobiveni rezultat jest najmanje potreban broj znamenaka pri čemu je potrebno uzeti u obzir da se logaritmiranjem dobiva decimalan broj. Njega je potrebno zaokružiti na manju cijelu vrijednost i uzeti u obzir da se broj

znamenaka dobiven logaritmiranjem broja koji je jednak cjelobrojnoj potenciji baze logaritmiranja mora povećati za 1. Općenito se izračun potrebnog broja znamenaka može dati sljedećim izrazom:

$$n = floor(log_b(broj_{10})) + 1, \tag{2.4}$$

gdje n označava najmanji broj znamenaka u novoj bazi, b bazu novog brojevnog sustava, $floor$ predstavlja funkciju zaokruživanja na manju cjelobrojnu vrijednost, a $broj_{10}$ najveću dekadsku vrijednost raspona koji se želi prikazati u novoj bazi.

2.1.1 Dekadski brojevni sustav

Dekadski brojevni sustav jest najprikladniji i najprirodniji za čovjeka jer mu je baza jednaka 10 što je pak jednako broju prstiju na obje ruke. Stoga je dekadski sustav danas najrasprostranjeniji brojevni sustav. Ovaj sustav također koristi položajni način zapisivanja brojeva. Za zapis broja u dekadskom brojevnom sustavu koriste se sljedeće znamenke:

$$\boldsymbol{Z} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}. \tag{2.5}$$

Radi lakšeg razumijevanja položajnih brojevnih sustava i kako se koriste težine pojedine znamenke najbolje je iskoristiti dekadski brojevni sustav na primjeru plaćanja. Znamo da su za plaćanje dostupne samo određene novčanice i da ukupni iznos slažemo kombinacijama pojedinih novčanica. Pretpostavimo da su nam samo dostupne novčanice s iznosom koji se može dobiti potenciranjem broja 10. Radi se o: (i) novčanici od 1.000 HRK (težina 10^3); (ii) novčanici od 100 HRK (težina 10^2); (iii) novčanici od 10 HRK (težina 10^1); (iv) kovanici od 1 HRK (težina 10^0); (v) kovanici od 10 lipa (težina 10^{-1}); i (vi) kovanici od 1 lipe (težina 10^{-2}). Potrebno je platiti iznos od 1.234,56 HRK, odnosno ako to prikažemo pomoću notacije definirane izrazom 2.2 dobivamo sljedeći prikaz:

$$P = 1234,56_{10} = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}. \tag{2.6}$$

Može se primijetiti kako je za plaćanje navedenog iznosa potrebno izdvojiti jednu novčanicu od 1.000 HRK, dvije novčanice od 100 HRK, tri novčanice od 10 HRK, četiri kovanice od 1 HRK, pet kovanica od 10 lipa te šest kovanica od 1 lipe. Druge nam kombinacije nisu dozvoljene budući da nisu dostupne druge novčanice, odnosno druge vrijednosti težina dobivene potenciranjem baze. Isti

se princip može primijeniti na položajne brojevne sustave s drugim bazama.

2.1.2 Binarni brojevni sustav

Kako je već navedeno, binarni brojevni sustav je prikladan za korištenje u računalima koja tijekom svog rada mogu razlikovati dva stanja (ima napona, nema napona). Time je automatski definirana baza binarnog brojevnog sustava, odnosno $b = 2$. To znači da binarni brojevni sustav koristi sljedeće znamenke:

$$\boldsymbol{Z} = \{0, 1\}. \tag{2.7}$$

I ovaj sustav je položajni brojevni sustav pa se vrijednost pojedinih težina može izraziti kao:

$$P = a_2 a_1 a_0, a_{-1} a_{-2}\ {}_2 = a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2}. \tag{2.8}$$

Položaj najznačajnije i najmanje značajne znamenke analogan je prethodno opisanom primjeru. Jedina je razlika što su sada pojedine težine jednake potenciji baze 2.

Prikaz cijelih pozitivnih brojeva

Kako ljudi koriste prikaz brojeva u bazi 10 potrebno je definirati postupak pretvorbe iz dekadskog u binarni brojevni sustav, odnosno iz dekadskog u bilo koji drugi brojevni sustav. Za pretvorbu cijelih brojeva koristi se metoda uzastopnog dijeljenja. Početni broj X u bazi 10 cjelobrojno se dijeli uzastopno s bazom drugog brojevnog sustava (u slučaju binarnog brojevnog sustava dijeli se s 2) tako dugo dok se kao rezultat cjelobrojnog dijeljenja ne dobije nula. Dobiveni ostaci cjelobrojnog dijeljenja predstavljaju znamenke broja prikazanog u drugoj bazi, pri čemu prvi ostatak predstavlja znamenku najmanje težine (samo je jednom dijeljeno s bazom prilikom pretvorbe), a zadnji ostatak najznačajniju znamenku (najviše je puta dijeljeno s bazom). To je najbolje vidljivo u sljedećem primjeru.

■ **Primjer 2.1**

Broj 28_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu pomoću metode uzastopnog dijeljenja.*

Rješenje:

Korištenjem gore opisane procedure moguće je napisati kako slijedi:

$$\begin{array}{lcll} 28:2 & = & 14 & ostatak\ 0 \quad najmanje\ značajna\ znamenka \\ 14:2 & = & 7 & ostatak\ 0 \\ 7:2 & = & 3 & ostatak\ 1 \\ 3:2 & = & 1 & ostatak\ 1 \\ 1:2 & = & 0 & ostatak\ 1 \quad najznačajnija\ znamenka \end{array} . \tag{2.9}$$

Poštujući pravilo o položaju najmanje i najviše značajne znamenke dobiveni rezultat možemo prikazati kao:

$$P = 28_{10} = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 11100_2. \tag{2.10}$$

Prilikom prikaza dekadskog broja u binarnom brojevnom sustavu postavlja se pitanje koja je najmanja količina bitova (binarnih znamenaka) potrebna za prikaz. Kao što je navedeno, ta informacija se dobiva logaritmiranjem dekadskog broja po bazi novog brojevnog sustava, zaokruživanjem rezultata na manju cjelobrojnu vrijednost i uvećavanje rezultata za 1. U ovom slučaju radi se o binarnom brojevnom sustavu, odnosno bazi 2. To se može onda odrediti sljedećim izrazom:

$$n = floor(log_2(broj_{10})) + 1, \tag{2.11}$$

pri čemu varijabla n označava traženi najmanji broj bitova, funkcija $floor$ zaokružuje svoj argument na manju cjelobrojnu vrijednost, a varijabla $broj_{10}$ označava dekadsku vrijednost koja se želi prikazati u binarnom brojevnom sustavu. Izračun potrebne količine bitova prikazan je u sljedećem primjeru 2.2.

■ **Primjer 2.2**

Parkiralište od 100_{10} *parkirnih mjesta želimo nadograditi sustavom za prikaz količine slobodnih parkirnih mjesta. Kako bi se to moglo ostvariti potrebno je mjeriti zauzeće svakog pojedinog parkirnog mjesta i uvesti sustav jednoznačne identifikacije svakog pojedinog parkirnog mjesta. To se radi pridruživanjem numeričke vrijednosti svakom parkirnom mjestu. Koja najmanja količina bitova je potrebna kako bi se svakom od* 100_{10} *parkirnih mjesta pridružila jednoznačna binarna vrijednost?*

Rješenje:

Opisani problem svodi se na izračun najmanjeg broja bitova potrebnih za prikaz danog dekadskog broja. Korištenjem izraza 2.11 moguće je napisati sljedeće:

$$n = floor(log_2(100)) + 1 = floor(6,64) + 1 = 6 + 1 = 7. \tag{2.12}$$

Kao rezultat dobivena je najmanja količina od 7 bitova. Sada se postavlja pitanje ispravnosti dobivenog rezultata odnosno nije li moguće ipak iskoristiti manju vrijednost od 6 bitova. Provjera dobivenog rezultata se može napraviti korištenjem izraza 2.3 za izračun raspona prikaza. Za dva razmatrana rješenja u slučaju binarnog brojevnog sustava moguće je napisati:

$$R_6 = [0, 2^6 - 1] = [0, 63], \tag{2.13}$$

$$R_7 = [0, 2^7 - 1] = [0, 127]. \tag{2.14}$$

Iz dobivenog raspona vidi se da pomoću 6 bitova možemo jednoznačno označiti najviše 64 parkirnih mjesta dok sa 7 bitova možemo jednoznačna označiti najviše 128 parkirnih mjesta. Ispravan rezultat jest količina od 7 bitova uz napomenu da će dio numeričkih vrijednosti (točnije njih 28) ostati neiskorišten, odnosno na raspolaganju za buduće proširenje parkirališta.

Brza pretvorba pozitivnih cijelih brojeva

Kao što je moguće vidjeti u izrazu 2.10, pozitivan cijeli broj u binarnom se brojevnom sustavu prikazuje pomoću težina dobivenih potencijom broja 2. Ta zakonitost se može iskoristiti za kreiranje brze metode pretvorbe pozitivnog dekadskog broja u binaran broj [9]. Metoda se zasniva na uzastopnom oduzimanju najveće moguće potencije broja 2 od početnog broja. Ostatak se prenosi u sljedeći korak u kojem se također oduzima najveća moguća potencija broja 2. Postupak se ponavlja tako dugo dok se ne dobije ostatak nula. Zbog ovog postupka pretvorbe ova metoda se naziva i metodom uzastopnog oduzimanja. Zatim se potencije broja 2, koje su korištene za oduzimanje, prikažu u obliku 2^x, pri čemu vrijednost x definira težinski položaj binarne vrijednosti 1 u konačnom binarnom broju. Težinske vrijednosti koje nisu korištene u oduzimanju imaju binarnu vrijednost 0. Ovaj postupak pretvorbe naročito je koristan kada se velike dekadske vrijednosti pretvaraju u binarnu vrijednost. Postupak je prikazan u sljedećem primjeru 2.3.

■ **Primjer 2.3**

Broj 28_{10} prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu pomoću metode brze pretvorbe.

Rješenje:

Korištenjem gore opisane procedure moguće je napisati kako slijedi:

$$\begin{array}{lcll} 28-16 & = & 12 & \textit{oduzeta vrijednost } 2^4 \\ 12-8 & = & 4 & \textit{oduzeta vrijednost } 2^3 \\ 4-4 & = & 0 & \textit{oduzeta vrijednost } 2^2 \\ & & & \textit{kraj postupka pretvorbe} \end{array} . \tag{2.15}$$

Pregledom vrijednosti koje su oduzete od početnog broja moguće je vidjeti da su korištene tri težinske vrijednosti. Njima će biti pridružena binarna vrijednost 1, a svim ostalim težinama binarna vrijednost 0. Prema tome možemo napisati:

$$P = 28_{10} = 1\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 = 1\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 = 11100_2. \tag{2.16}$$

Prikaz negativnih binarnih cijelih brojeva bitom predznaka

U slučaju da se žele prikazati i negativni cijeli brojevi potrebno je definirati način prikaza predznaka. Kod dekadskog brojevnog sustava koristi se ispred broja znak "+" za prikaz pozitivnog broja te znak "−" za prikaz negativnog broja. Pri tome se znak za pozitivni broj ne piše, već se pretpostavlja da je broj pozitivan. Binarni brojevni sustav se koristi u računalu, a vrijednosti se spremaju u potrebnu količinu bajtova. Tako se ne mogu koristiti klasični znakovi za predznak poznati iz matematike, već se jedan bit definira za prikaz predznaka. Uobičajeno je to najznačajniji bit i on tada ne doprinosi vrijednosti broja, već samo definira predznak. To je moguće vidjeti na slici 2.1 gdje je prikazan utjecaj bita za prikaz predznaka u slučaju da se broj prikazuje pomoću jednog bajta. Vrijednost 0 najznačajnijeg bita označava pozitivan predznak, a vrijednost 1 negativan predznak. Raspon koji možemo u ovom slučaju prikazati pomoću n bitova se može prikazati sljedećim izrazom:

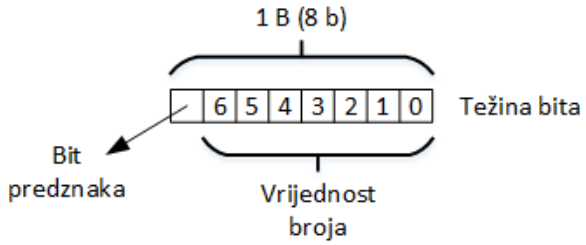
$$R = [-b^{n-1}+1, b^{n-1}-1]. \tag{2.17}$$

Uzmimo li kao primjer 3 bita tada se prema izrazu 2.17 može prikazati raspon od -3 do $+3$. U tablici 2.1 je dan pregled binarno kôdiranog broja, koji koristi najznačajniji bit za prikaz predznaka, i pripadne dekadske vrijednosti. Moguće je primijetiti da dva ista binarna kôda (000 i 100) označavaju u biti

istu vrijednost (+0 i −0). Time se nepotrebno smanjuje raspon brojeva koje je moguće prikazati. To je jedan od nedostataka ovog načina prikaza negativnih cijelih brojeva.

Binarni kôd	Dekadska vrijednost
000	+0
001	+1
010	+2
011	+3
100	-0
101	-1
110	-2
111	-3

Tablica 2.1: Primjer prikaza negativnog broja bitom predznaka u slučaju 3 bita.



Slika 2.1: Prikaz negativnih binarnih brojeva pomoću bita za predznak.

Dodatni nedostatak ovog načina prikaza predznaka je što se jedan bit gubi za prikaz predznaka i dobiveni kôd broja nije pogodan za izravnu obradu u aritmetičko-logičkoj jedinici mikroprocesora. To znači da nije moguće izravno zbrojiti istu pozitivnu i negativnu vrijednost te kao rezultat dobiti vrijednost nula. Primjer pretvorbe dan je u primjeru 2.4.

■ **Primjer 2.4**

Broj -4_{10} prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu pomoću bita za predznak najmanjim brojem bitova. Objasnite odabir broja bitova za prikaz.

Rješenje:

U ovom postupku pretvorbe potrebno je prvo pretvoriti pripadni pozitivan

dekadski broj u binaran. Korištenjem brze metode pretvorbe možemo pisati:

$$P = 4_{10} = 2^2 = 1 \cdot 2^2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 100_2. \tag{2.18}$$

Prema dobivenom međurezultatu danim izrazom 2.18 moguće je zaključiti da je za prikaz pripadnog pozitivnog broja potrebno najmanje 3 bita. Budući da se prikazuje predznak, potrebno je dodati još jedan bit. To znači da je za ispravan prikaz u ovom slučaju potrebno najmanje 4 bita pri čemu najznačajniji bit ima samo funkciju prikaza predznaka. On ne doprinosi vrijednosti broja. Konačno možemo pisati za cijelu pretvorbu:

$$P = -4_{10} = -2^2 = -1 \cdot 2^2 = -1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -100_2 = 1100_2. \tag{2.19}$$

Prikaz negativnih binarnih brojeva dvojnim komplementom

Kako bi se negativni brojevi mogli izravno obrađivati u aritmetičko-logičkoj jedinici u mikroprocesoru, definiran je prikaz negativnih binarnih brojeva pomoću dvojnog komplementa. U ovom slučaju je moguće izravno zbrojiti istu pozitivnu i negativnu vrijednost te kao rezultat dobiti vrijednost nula. Pretvorba negativnog dekadskog broja u prikaz dvojnim komplementom provodi se na način da se prvo pretvori pripadna pozitivna (apsolutna) vrijednost u binarni broj postupkom koji je opisan ranije. Zatim se radi komplement svakog pojedinog bita, odnosno trenutna vrijednost pojedinog bita se pretvori u njegovu suprotnu vrijednost (vrijednost 0 postaje vrijednost 1 i obrnuto). Na kraju se tako dobivenom binarnom broju još zbrajanjem doda binarna vrijednost 1. Cijeli postupak je prikazan primjerom 2.5.

Prilikom pretvorbe potrebno je obratiti pažnju koliko bajtova se koristi za prikaz negativnog broja. Dobiveni pozitivni binarni broj mora se moći prikazati bajtovima koji su na raspolaganju uz uvjet da najznačajniji bit ima vrijednost nula. To znači da je za prikaz pozitivnog broja dostupno ukupno $m \cdot 8 - 1$ bitova, pri čemu m označava količinu bajtova dostupnih za prikaz. U slučaju da je za prikaz pozitivnog broja dostatan manji broj bitova, svim značajnijim bitovima se pridružuje vrijednost 0. Najznačajniji bit i u ovom prikazu definira predznak broja. Analogno prikazu pomoću bita za predznak i u ovom slučaju vrijedi da ukoliko je vrijednost najznačajnijeg bit jednaka 0 imamo pozitivan broj, odnosno ukoliko je njegova vrijednost 1 imamo negativan broj.

Raspon vrijednosti koji je moguće prikazati i u ovom slučaju ovisi o količini memorije odnosno bajtova ili bitova dostupnih za prikaz. Taj raspon je malo veći nego u slučaju prethodnog prikaza negativnih brojeva i iznosi:

$$R = [-b^{n-1}, b^{n-1} - 1]. \tag{2.20}$$

Do nešto većeg raspona prikaza dolazi iz razloga što se vrijednost nula ne prikazuje pomoću dva različita binarna kôda, a razliku je moguće vidjeti u tablici 2.2 na primjeru prikaza negativnih brojeva pomoću 3 bita. U tablici 2.2 moguće je primijetiti još jednu značajku dvojnog komplementa. Ako se najvećem pozitivnom broju doda jedan dobiti će se najmanji negativni broj. Prema tablici 2.2 za slučaj 3 bita najveći pozitivni broj iznosi 3 s binarnim kôdom 011. Dodamo li mu jedan dobiti ćemo binarni kôd 100 što je prema tablici 2.2 dekadska vrijednost -4. Ista značajka postoji i u računalima kada se ovaj format koristi za prikaz cijelih brojeva s predznakom. Iz tog razloga je prilikom pisanja programa potrebno posvetiti pažnju procjeni raspona vrijednosti pojedinih varijabli kako bi se odabrali ispravni tipovi za prikaz podataka.

Binarni kôd	Dekadska vrijednost
000	+0
001	+1
010	+2
011	+3
100	-4
101	-3
110	-2
111	-1

Tablica 2.2: Primjer prikaza negativnog cijelog broja dvojnim komplementom u slučaju 3 bita.

■ **Primjer 2.5**

Broj -28_{10} prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu pomoću dvojnog komplementa. Za prikaz je potrebno iskoristiti najmanji potreban broj bajtova.

Rješenje:

Pripadni pozitivan broj odnosno apsolutna vrijednost negativnog broja -28_{10} je 28_{10}. U primjeru 2.1 je taj broj već pretvoren u pripadni binarni broj što iznosi 11100_2. Za prikaz pripadnog pozitivnog broja 28_{10} je potrebno najmanje 5 bitova što znači da je za prikaz pripadnog negativnog broja potrebno najmanje 6 bitova. Kako je memorija organizirana po bajtova dobivamo da je za prikaz

negativnog broja dovoljan jedan bajt. Proširivanje prikaza pozitivnog broja 28_{10} na jedan bajt dobivamo vrijednost 00011100_2. Najznačajniji bit ima vrijednost nula što označava pozitivan broj.

Sljedeći korak nalaže određivanje komplementa svakog bita što kao rezultat daje vrijednost 11100011_2. Dodavanjem vrijednost 1 konačno se dobiva vrijednost 11100100_2 što predstavlja rezultat ovog primjera. Najznačajniji bit ima vrijednost 1 što označava negativan broj. U slučaju da se dobiveni prikazi pozitivnog i negativnog broja zbroje, kao rezultat se dobiva vrijednost 00000000_2 uz preljev 1_2. Upravo ova značajka omogućuje izravne aritmetičke operacije unutar aritmetičko-logičke jedinice mikroprocesora. Tako je dvojni komplement osnova za prikaz cijelih brojeva s predznakom u računalu. Preljev pri tome znači da se dobivena vrijednost više ne može prikazati u količini memorije dostupnoj za spremanje broja. U slučaju sume istog pozitivnog i negativnog broja se preljev detektira i zanemaruje jer se radi o posebnom slučaju.

Pretvorba decimalnog dijela brojeva

U tehničkim sustavima se za prikaz različitih vrijednosti puno češće od cijelih brojeva koriste decimalni brojevi. Oni se sastoje od dva dijela odvojenih decimalnim zarezom. Prvi dio prikazuje cijeli dio broja dok drugi dio prikazuje decimalni dio broja. Prilikom pretvorbe decimalnog dekadskog broja u binarni broj navedeni se dijelovi pretvaraju odvojeno. Prvo se pretvara cijeli dio broja opisanim postupkom. Zatim se pretvara decimalni dio broja i to na način da se on pomnoži s brojem dva, odnosno bazom binarnog brojevnog sustava. Cijeli dio dobivenog umnoška predstavlja jednu novu binarnu znamenku. Za sljedeće množenje se dalje uzima samo decimalni dio broja. Postupak se ponavlja tako dugo dok se kao rezultat umnoška ne pojavi nula, odnosno dok se ne potroše raspoloživi bitovi za prikaz decimalnog dijela broja. Drugi uvjet prestanka je potreban iz razloga što se ne mogu svi decimalni brojevi točno prikazati u binarnom brojevnom sustavu. Na kraju se iza decimalnog zareza prepišu dobivene cijele znamenke iz svakog koraka množenja uključujući i nule. Redoslijed prepisivanja je takav da je najznačajnija znamenka rezultat prvog množenja. Ostale znamenke idu redom prema pojedinom koraku množenja. Primjer 2.6 prikazuje postupak pretvorbe decimalnog dijela dekadskog broja u binarni dok je u primjeru 2.7 prikazana pretvorba decimalnog dekadskog broja kojeg nije moguće točno prikazati binarnim brojem.

Potrebno je obratiti pažnju da engleski pravopis i literatura te programski jezici koriste decimalnu točku za razliku od decimalnog zareza koji koristimo u hrvatskom pravopisu. Ova razlika može dovesti do zabune i krivog unosa prilikom razmjene podataka između pojedinih programa i aplikacija. Iz tog

razloga je prije unosa podataka potrebno napraviti provjeru postavki računala vezani uz odabranu državu korisnika.

■ **Primjer 2.6**

Broj $0,125_{10}$ *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu.*

Rješenje:

Korištenjem gore opisane procedure pretvorbe moguće je napisati kako slijedi:

$$\begin{array}{lcll} 0,125 \cdot 2 & = & 0,25 & \textit{cijeli dio } 0 \quad \textit{najznačajnija znamenka} \\ 0,25 \cdot 2 & = & 0,5 & \textit{cijeli dio } 0 \\ 0,5 \cdot 2 & = & 1,0 & \textit{cijeli dio } 1 \quad \textit{najmanje značajna znamenka} \end{array} . \quad (2.21)$$

Poštujući pravilo o položaju najmanje i najviše značajne znamenke dobiveni rezultat se može prikazati kao:

$$P = 0,125_{10} = 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0,001_2. \quad (2.22)$$

■ **Primjer 2.7**

Broj $0,6_{10}$ *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu. Komentirajte je li je mogući ovaj broj točno prikazati u binarnom brojevnom sustavu!*

Rješenje:

Korištenjem gore opisane procedure pretvorbe moguće je napisati kako slijedi:

$$\begin{array}{lcll} 0,6 \cdot 2 & = & 1,2 & \textit{cijeli dio } 1 \quad \textit{najznačajnija znamenka} \\ 0,2 \cdot 2 & = & 0,4 & \textit{cijeli dio } 0 \\ 0,4 \cdot 2 & = & 0,8 & \textit{cijeli dio } 0 \\ 0,8 \cdot 2 & = & 1,6 & \textit{cijeli dio } 1 \\ 0,6 \cdot 2 & = & 1,2 & \textit{cijeli dio } 1 \quad \textit{dobiven početni broj} \end{array} . \quad (2.23)$$

U izrazu 2.23 moguće je primijetiti kako se u petom koraku pretvorbe ponovno javlja početni broj $0,6_{10}$. Ponavljanje procedure pretvorbe više nema smisla jer će se početni broj ponovno pojaviti. Binarni kôd 1001 se ponavlja što nameće zaključak da broj $0,6_{10}$ nije moguće točno prikazati u binarnom brojevnom sustavu. Točnost prikaza ovisi o količini raspoložive memorije. Iz tog razloga je moguće da se pojave problemi prilikom obrade numeričkih podataka u računalu. Analogija se može napraviti s prikazom razlomka 1/3 kojeg također nije moguće točno prikazati decimalnim brojem u bazi 10. Točnost prikaza ovisi o broju korištenih decimala.

Poštujući pravilo o položaju najmanje i najviše značajne znamenke dobiveni rezultat se može prikazati kao:

$$P = 0,6_{10} = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + ... = 0,10011001..._2. \quad (2.24)$$

2.1.3 Oktalni brojevni sustav

Oktalni brojevni sustav također predstavlja položajni brojevni sustav s bazom jednakom osam, odnosno $b = 8$. Prema definiciji oktalni brojevni sustav sadrži sljedeće znamenke:

$$\boldsymbol{Z} = \{0,1,2,3,4,5,6,7\}. \quad (2.25)$$

Koristi se za jednostavniji, odnosno skraćeniji prikaz binarnog broja. To je vrlo korisno kod pregleda različitih binarnih datoteka u računalu kada je potrebno detaljnije analizirati zapise u takvim datotekama. Binarne datoteke, za razliku od tekstualnih, sadrže zapise razumljive računalu, odnosno pripadnom programu koji ih je generirao. Za lakše prevođenje binarnih podataka u oktalni brojevni sustav i obrnuto moguće je iskoristiti tablicu 2.3. Pretvorba između dekadskog i oktalnog brojevnog sustava radi se analogno pretvorbi kod binarnog brojevnog sustava. Postupak pretvorbe dan je primjerom 2.8.

■ **Primjer 2.8**

Broj 28_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u oktalnom brojevnom sustavu pomoću metode uzastopnog dijeljenja.*

Rješenje:

Korištenjem gore opisane procedure moguće je napisati kako slijedi:

Dekadska vrijednost	Binarni kôd	Oktalna znamenka
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

Tablica 2.3: Usporedni prikaz dekadske vrijednosti te pripadnog binarnog kôda i oktalne znamenke.

$$\begin{array}{lcl} 28:8 & = & 3 \quad ostatak \quad 4 \quad najmanje\ značajna\ znamenka \\ 3:8 & = & 0 \quad ostatak \quad 3 \quad najznačajnija\ znamenka \end{array} . \tag{2.26}$$

Poštujući pravilo o položaju najmanje i najviše značajne znamenke dobiveni rezultat se može prikazati kao:

$$P = 28_{10} = 3 \cdot 8^1 + 4 \cdot 8^0 = 34_8. \tag{2.27}$$

2.1.4 Heksadecimalni brojevni sustav

Heksadecimalni brojevni sustav je položajni brojevni sustav s iznosom baze, $b = 16$. Ukupno ima 16 različitih znamenaka pri čemu su dekadske vrijednosti od 10 do 15 označene velikim slovima od A do F. Slovčane oznake se koriste iz razloga što se dekadske vrijednosti od 10 do 15 prikazuju kombinacijama dviju već iskorištenih znamenki heksadecimalnog brojevnog sustava. Uvođenje slovčanih oznaka omogućuje jednoznačni prikaz znamenaka. Pregled svih korištenih znamenaka heksadecimalnog sustava i njihovih pripadnih dekadskih vrijednosti, odnosno binarnih kôdova dan je u tablici 2.4. Kao oznaka baze broja danog u heksadecimalnom formatu uz klasičnu oznaku baze 16 koristi se i oznaka h. Tako se kao primjer oznake heksadecimalnog broja može navesti $CAFFE_{16}$ odnosno istovjetni $CAFFE_h$. Pretvorba između dekadskog i heksadecimalnog brojevnog sustava radi se analogno pretvorbi kod binarnog brojevnog sustava. Postupak pretvorbe dan je primjerom 2.9.

■ **Primjer 2.9**

Dekadska vrijednost	Binarni kôd	Heksadecimalna znamenka
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Tablica 2.4: Usporedni prikaz dekadske vrijednosti te pripadnog binarnog kôda i heksadecimalne znamenke.

Broj 28_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u heksadecimalnom brojevnom sustavu pomoću metode uzastopnog dijeljenja.*

Rješenje:

Korištenjem gore opisane procedure moguće je napisati kako slijedi:

$$\begin{array}{lcll} 28:16 & = & 1 \;\; ostatak \;\; 12 & najmanje\ značajna\ znamenka \\ 1:16 & = & 0 \;\; ostatak \;\; 1 & najznačajnija\ znamenka \end{array} . \tag{2.28}$$

Poštujući pravilo o položaju najmanje i najviše značajne znamenke dobiveni rezultat se može prikazati kao:

$$P = 28_{10} = 1 \cdot 16^1 + 12 \cdot 16^0 = 1 \cdot 16^1 + C \cdot 16^0 = 1C_{16}. \tag{2.29}$$

2.1.5 Binarno kôdirano decimalni (BCD) brojevi

Ljudi kao prirodnu bazu za prikaz brojeva koriste bazu 10. Razlog je što na rukama ljudi imaju 10 prstiju što je u doba antike olakšavalo ručni izračun prije pojave prve mehaničke pomoći za računanje, abakusa. Kao što je već navedeno, za prikaz brojeva korištenjem baze 10 potrebno je moći prikazati 10 različitih znamenaka od 0 do 9. Usporedimo li broj znamenaka navedenih brojevnih sustava koji omogućuju prikaz podataka u računalima, moguće je zaključiti da jedino heksadecimalni brojevni sustav može prikazati potreban broj znamenaka. Pri tome postoji problem što heksadecimalni brojevni sustav ima 16 različitih znamenaka. Za prikaz brojeva prikazanih u BCD formatu se koristi samo dio znamenaka heksadecimalnog brojevnog sustava dok ostale znamenke predstavljaju nedopušteni kôd. Pregled znamenaka te pripadnih dekadskih vrijednosti kao i pripadni binarni kôd dan je u tablici 2.5.

Dekadska vrijednost	Binarni kôd	BCD znamenka
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

Tablica 2.5: Usporedni prikaz dekadske vrijednosti te pripadnog binarnog kôda i BCD znamenke.

Pretvorba dekadskog broja u BCD format radi se tako da se prvo svaka znamenka dekadskog broja zasebno prikaže svojim pripadnim BCD kôdom. Pri tome je za prikaz svake znamenke potrebno koristiti 4 bita. Ukoliko se znamenka može prikazati manjim brojem bitova, značajniji bitovi se izjednače s nulom kako bi se dobio prikaz pomoću 4 bita. Zbog ovakve pretvorbe više se ne radi o položajnom brojevnom sustavu. BCD format koristi se za prikaz numeričkih podataka u komunikaciji između računala i operatera iz razloga što ljudi prirodno razumiju samo numeričke podatke prikazane u bazi 10. Za brojeve prikazane u BCD formatu ne postoji posebna oznaka pa će se u ovoj skripti koristiti oznaka BCD. Pregled postupka pretvorbe dekadskog broja u BCD format

s pripadnim binarnim kôdom za prikaz u računalu dan je u primjeru 2.10.

■ **Primjer 2.10**

Broj 28_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u BCD brojevnom formatu.*

Rješenje:

Prema opisanoj proceduri pretvorbe potrebno je zasebno svaku dekadsku znamenku prikazati u binarnom brojevnom sustavu korištenjem 4 bita. Moguće je prema tome napisati kako slijedi za prvu znamenku:

$$\begin{array}{lcll} 2:2 & = & 1 \quad ostatak \quad 0 & najmanje\ značajna\ znamenka \\ 1:2 & = & 0 \quad ostatak \quad 1 & najznačajnija\ znamenka \end{array}, \tag{2.30}$$

odnosno za drugu znamenku vrijedi:

$$\begin{array}{lcll} 8:2 & = & 4 \quad ostatak \quad 0 & najmanje\ značajna\ znamenka \\ 4:2 & = & 2 \quad ostatak \quad 0 & \\ 2:2 & = & 1 \quad ostatak \quad 0 & \\ 1:2 & = & 0 \quad ostatak \quad 1 & najznačajnija\ znamenka \end{array}. \tag{2.31}$$

Poštujući pravilo o položaju najmanje i najviše značajne znamenke kod pretvorbe u binarni brojevni sustav s prikazom pomoću 4 bita dobiveni rezultati pretvorbe znači da za prvu znamenku vrijedi:

$$P = 2_{10} = 10_2 = 0010_2, \tag{2.32}$$

dok za drugu znamenku vrijedi:

$$P = 8_{10} = 1000_2. \tag{2.33}$$

Ukupni rezultat pretvorbe sada se može prikazati kao:

$$P = 28_{10} = 0010_2\ 1000_2 = 28_{BCD}. \tag{2.34}$$

Moguće je primijetiti da je po prikazu znamenki BCD broj jednak deka-

dskom, a razlika je u prikazu u binarnom formatu. Razlika u binarnom prikazu se javlja iz razloga što ovaj format nije položajni, već svaka znamenka definira po 4 bita binarnog prikaza.

2.2 Pretvorba između brojevnih sustava

Radi pojednostavljenja pretvorbe između pojedinih brojevnih sustava moguće je iskoristiti vezu između osnovnih, dekadskog i binarnog brojevnog sustava, te ostalih brojevnih sustava. Veza se uspostavlja preko činjenice da je npr. za prikaz bilo koje znamenke oktalnog brojevnog sustava u binarnom brojevnom sustavu potrebno 3 bita. Za prikaz bilo koje heksadecimalne znamenke u binarnom brojevnom sustavu potrebno je 4 bita. Tako je za pretvorbu između navedenih brojevnih sustava bitno poznavati pretvorbu iz dekadskog u binarni brojevni sustav te prikaz pojedinih znamenaka oktalnog, heksadecimalnog te BCD brojevnog sustava u binarnom brojevnom sustavu. Potrebna pretvorba između binarnog te oktalnog, heksadecimalnog i BCD brojevnog sustava dana je u tablicama 2.3, 2.4 i 2.5.

Procedura pretvorbe razlikuje se u ovisnosti tome o kojem smjeru se radi. Ako je početni broj dan u dekadskom brojevnom sustavu tada je prvi korak pretvorba iz dekadskog formata u binarni. Sljedeći korak ovisi o tome koji je sljedeći brojevni sustav. Za pretvorbu iz binarnog u oktalni brojevni sustav potrebno je grupirati binarne znamenke u skupine od po tri znamenke. Pri tome grupiranje počinje kod najmanje značajne znamenke lijevo uz decimalni zarez. Ako je broj najznačajnijih bitova manji od tri tada se svi značajniji bitovi izjednačavaju s nulom. Nakon toga se svaka skupina od tri bita pretvori u pripadnu oktalnu znamenku. Procedura kod heksadecimalne znamenke je analogna s time da se bitovi grupiraju u skupine od četiri bita. Ako se radi o pretvorbi u suprotnom smjeru, tada se u prvom koraku svaka oktalna ili heksadecimalna znamenka zamijeni sa skupinom od tri odnosno četiri bita. Dobivene grupe bitova se spajaju u binaran broj i uz korištenje pravila položajnog brojevnog sustava pretvore u dekadski broj. Sama procedura pretvorbe prikazana je u primjerima 2.11 i 2.12.

■ **Primjer 2.11**

Broj 75_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u oktalnom brojevnom sustavu korištenjem binarnog brojevnog sustava.*

Rješenje:

Prvi korak pretvorbe je pretvaranja dekadskog broja 75_{10} u binarni broj

korištenjem metode uzastopnog dijeljenja. Njezinom primjenom dobiva se:

$$\begin{array}{lcll} 75:2 & = & 37 \quad ostatak \quad 1 & najmanje\ značajna\ znamenka \\ 37:2 & = & 18 \quad ostatak \quad 1 & \\ 18:2 & = & 9 \quad ostatak \quad 0 & \\ 9:2 & = & 4 \quad ostatak \quad 1 & \\ 4:2 & = & 2 \quad ostatak \quad 0 & \\ 2:2 & = & 1 \quad ostatak \quad 0 & \\ 1:2 & = & 0 \quad ostatak \quad 1 & najznačajnija\ znamenka \end{array} . \tag{2.35}$$

Dobiveni binarni broj 1001011_2 se u drugom koraku grupira u grupe od po tri bita koje se zatim korištenjem tablice 2.3 pretvaraju u pripadne oktalne znamenke čime se dobiva konačni rezultat pretvorbe:

$$P = 75_{10} = 1001011_2 = 1\ 001\ 011_2 = 001\ 001\ 011_2 = 113_8. \tag{2.36}$$

■ **Primjer 2.12**

Broj $A10_{16}$ prikazan u heksadecimalnom brojevnom sustavu potrebno je prikazati u dekadskom brojevnom sustavu korištenjem binarnog brojevnog sustava.

Rješenje:

Kako bi se napravila ova pretvorba, prvo se sve heksadecimalne znamenke pretvaraju u pripadni binarni broj korištenjem tablice 2.4. Pri tome se poštuje pravilo da se svaka heksadecimalna znamenka prikazuje s četiri bita. Dobiveni binarni broj spoji se u jednu grupu koja se zatim prema težini svakog pojedinog bita pretvara u dekadski broj. Na kraju procedure dobiva se konačni rezultat pretvorbe:

$$\begin{array}{rcl} P & = & A10_{16} \\ & = & 1010\ 0001\ 0000_2 \\ & = & 101000010000_2 \\ & = & 1 \cdot 2^{11} + 1 \cdot 2^9 + 1 \cdot 2^4 \\ & = & 2576_{10} \end{array} . \tag{2.37}$$

Radi jednostavnosti prikaza u koraku pretvorbe iz binarnog u dekadski broj

ispuštena su množenja onih težina kojima je pridružen bit s vrijednošću nula.

2.3 Zadaci za samostalan rad

U nastavku se nalazi skup zadataka za samostalan rad. Za rješavanje zadataka preporuča se iskoristiti već riješene primjere te teorijsku podlogu danu u ovom poglavlju. Kao pomoć može poslužiti i program "*Calculator*" ugrađen u operacijski sustav MS Windows koji u "*Programmer*" načinu prikaza podržava u ovoj skripti opisane brojevne sustave. Također omogućava pretvorbu između pojedinih brojevnih sustava.

■ Zadatak 2.1

Broj 1234_{10} *prikazan u dekadskom brojevnom sustavu potrebno je prikazati u binarnom brojevnom sustavu. Koliko je najmanje bitova, odnosno bajtova potrebno kako bi se dobiveni binarni broj mogao spremiti u memoriju računala?*

■ Zadatak 2.2

Broj $0,56_{10}$ *prikazan u dekadskom brojevnom sustavu prikažite u binarnom brojevnom sustavu. Jeli navedeni broj moguće točno prikazati u binarnom brojevnom sustavu? Objasnite odgovor!*

■ Zadatak 2.3

Broj -15_{10} *prikazan u dekadskom brojevnom sustavu prikažite u binarnom brojevnom sustavu korištenjem bita za prikaz predznaka. Koliko je najmanje bitova potrebno kako bi se ovaj broj mogao prikazati pomoću dvojnog komplementa? Objasnite odgovor!*

■ Zadatak 2.4

Broj -9_{10} *prikazan u dekadskom brojevnom sustavu prikažite u binarnom brojevnom sustavu korištenjem dvojnog komplementa. Koliko je najmanje bitova potrebno kako bi se ovaj broj mogao prikazati pomoću dvojnog komplementa? Objasnite odgovor!*

■ Zadatak 2.5

Broj -149_{10} *prikazan u dekadskom brojevnom sustavu prikažite u binarnom*

brojevnom sustavu korištenjem metode brze pretvorbe, prikaza bitom za predznak i dvojnog komplementa. Objasnite značenje pojedinog bita u dobivenim prikazima!

■ Zadatak 2.6

Koliki je raspon brojeva moguće prikazati pomoću 2 *bajta korištenjem sljedećih prikaza:*

a) Binarni brojevni sustav;
b) Binarni brojevni sustav s bitom za predznak;
c) Prikaz pomoću dvojnog komplementa.

Objasnite odgovor za svaki prikaz!

■ Zadatak 2.7

Broj 19_{10} *prikazan u dekadskom brojevnom sustavu prikažite u oktalnom brojevnom sustavu. Ako oba broja predstavljaju starost čovjeka odnosno broj godina, koju bazu je bolje uzeti? Objasnite odgovor!*

■ Zadatak 2.8

Broj 38_{10} *prikazan u dekadskom brojevnom sustavu prikažite u heksadecimalnom brojevnom sustavu. Ako oba broja predstavljaju starost čovjeka odnosno broj godina, koju bazu je bolje uzeti? Objasnite odgovor!*

■ Zadatak 2.9

Broj 47_8 *prikazan u oktalnom brojevnom sustavu prikažite u heksadecimalnom brojevnom sustavu. Objasnite odabrani način pretvorbe!*

■ Zadatak 2.10

Broj -39_{16} *prikazan u heksadecimalnom brojevnom sustavu prikažite pomoću dvojnog komplementa. Koliko je najmanje bajtova potrebno za prikaz broja u dvojnom komplementu? Objasnite odgovor!*

Poglavlje 3

Zauzeće memorije i prikaz podataka

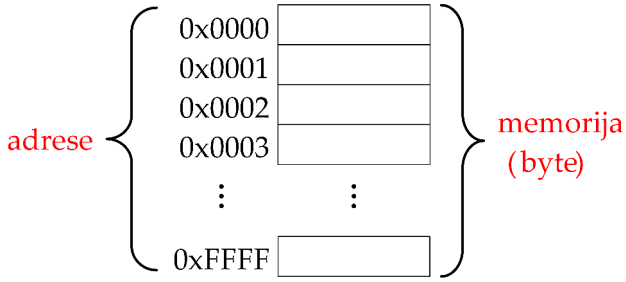
Računala za prikaz svih vrsta podataka koriste numeričke kôdove, odnosno osnova prikaza je binarni brojevni sustav. Ostali brojevni sustavi s većom bazom (oktalni i heksadecimalni) i BCD kôd služe za kraći prikaz korištenog binarnog kôda, odnosno za olakšanje komunikacije između čovjeka (operatera) i računala. U ovom poglavlju opisati će se na koji način se podaci spremaju u memoriju računala.

3.1 Organizacija memorije

U uvodnom poglavlju već je spomenuta memorija kao jedan od ključnih elemenata građe računala. Memorija je sastavljena od elektroničkih čipova koji služe za spremanje pojedinih bitova. Kako je bit vrlo mala jedinica, bitovi se grupiraju u skupine od 8 bitova odnosno bajtove (oktet ili slovnjak, engl. "byte"). Bajt čini najmanji dio memorije kojem se može pristupiti.

Pristup pojedinom bajtu memorije radi se korištenjem njegove adrese. Pri tome adresa bajta predstavlja jednoznačan broj koji je pridružen svakom bajtu kako bi se moglo pristupiti njegovom sadržaju. Prvi bajt ima adresu jednaku 0, a adresa svakog sljedećeg bajta je veća za jedan. Primjer adresiranja memorije prikazan je slikom 3.1. Ovisno o tipu podatka koji se prikazuje potrebna je različita količina memorije, odnosno broj bajtova. Najčešće je ta količina veća od jednog bajta pa se radi grupiranje bajtova. Dobivena grupa bajtova predstavlja sada memorijsku lokaciju podatka i ona se također adresira pri čemu se

kao adresa cijele grupe bajtova (memorijske lokacije) uzima adresa prvog bajta. Radi lakše manipulacije s podacima u memoriji računala, cijeli proces je automatiziran i provodi ga operacijski sustav. Bitno je samo u programu ispravno odabrati tip podataka.



Slika 3.1: Adresiranje pojedinog bajta memorije u računalu [12].

3.1.1 Mjerenje količine memorije

Bitno je napomenuti da se adresa memorijske lokacije također prikazuje binarnim brojem kako je vidljivo na slici 3.1. Radi jasnijeg prikaza na slici 3.1 je korišten heksadecimalni brojevni sustav za oznaku adrese svakog pojedinog bajta. Kako bi se adresa pojedinog bajta razlikovala od drugih numeričkih podataka, ona uvijek počinje s oznakom $0x$. Količina memorije mjeri se korištenjem zakonitosti binarnog brojevnog sustava, odnosno koristi se baza 2. Za mjerenje količine memorije koristi se veća jedinica od bita, odnosno bajt (oznaka B). Kako današnja računala imaju u sebe ugrađene velike količine memorije, koriste se prefiksi osnovne jedinice. Sustav prefiksa u bazi 2 je analogan sustavu prefiksa koji je poznat u bazi 10. Radi se o prefiksima kilo (oznaka k, vrijednost 10^3), mega (oznaka M, vrijednost 10^6) itd. Tako se u svakodnevnoj komunikaciji koriste vrijednosti poput kilobajta (oznaka kB, količina od 10^3B), megabajt (oznaka MB, količina od 10^6B) itd.

Problem u korištenju ovih prefiksa je što postoji mala razlika prilikom definiranja analognih prefiksa u dekadskom sustavu i u binarnom sustavu. Naime, vrijednost u binarnom sustavu analogna prefiksu kilo iz dekadskog brojevnog sustava jest kibi, vrijednosti 2^{10} odnosno 1024. Vidljivo je da postoji razlika u vrijednosti koja iznosi $24B$ odnosno $2,4\%$ od ukupne količine. To nije tako velika razlika, no ta razlika je veća sa svakim sljedećim prefiksom kako je vidljivo u tablici 1.2. U slučaju $1GB$ razlika već iznosi $7,37\%$ što više nije zanemarivo. Za veće količine memorije kao što su terabajti (TB) razlika već doseže $9,95\%$. Zbog jednostavnosti se za prikaz količine memorije u svakodnevnom korištenju

izjednačavaju prefiksi dekadskog i binarnog brojevnog sustava. Iz ovog razloga dešava se da je stvarna količina memorije tvrdog diska izražena u bajtovima manja od one deklarirane od strane proizvođača. Isti način označavanja će se koristiti u sklopu ove skripte kako je to uvriježena praksa svih proizvođača.

3.1.2 Manipulacija podacima u memoriji

Svi podaci u računalu spremljeni su u nekoj memoriji. Tijekom izvršavanja programa podaci iz trajne memorije (tvrdi disk, DVD-disk i sl.) prebacuju se u radnu memoriju kako bi se ubrzala obrada podataka. Podacima se pristupa na način da se navede početna adresa podatka (adresa prvog bajta), tip podatka i broj bajtova u kojima je spremljena informacija. Kada se podaci dohvaćeni iz memorije koriste tijekom izvršavanja programa, tip podatka je vrlo bitna informacija kako on opisuje način na koji se interpretira značenje pojedinog bita.

Pristup podacima pomoću varijabli

Navođenje točne adrese odnosno adrese prvog bajta predstavlja dosta složen način pristupanja podacima u memoriji. Kako bi se pristup podacima olakšao, koriste se varijable odnosno simboličke oznake memorijskih lokacija sa spremljenim podacima. Varijable predstavljaju svaku vrijednost koja se mijenja tijekom izvođenja programa. Pri tome vrijednost treba biti iz skupa dopuštenih vrijednosti varijable. Taj skup vrijednosti je definiran tipom podatka koji se sprema u varijablu.

Prvi korak u korištenju varijabli je njihova definicija odnosno deklaracija. Ovisno o programskom jeziku koriste se posebne naredbe ili se samo navede tip i ime varijable. Deklaracijom varijable zauzima se potrebna memorija i stvara se simbolička poveznica sa zauzetom memorijom. Nakon deklaracije varijable, podacima u memoriji moguće je pristupiti navođenjem danog imena varijable. Operacijski sustav automatski dohvaća adresu varijable i njen sadržaj (sam podatak) prenosi iz memorije u akumulator mikroprocesora radi daljnje obrade.

Prilikom zadavanja imena varijable potrebno je poštivati određena pravila i preporuke. Pravila koja se uvijek moraju poštivati prilikom zadavanja imena varijable jesu:

- Dozvoljeno je koristiti samo slova engleske abecede;
- Nije dozvoljeno koristiti ključne riječi pojedinog programskog jezika;
- Prvi znak u imenu varijable mora biti slovo;
- Svaka varijabla mora imati svoje jednoznačno ime;

- Velika i mala slova se razlikuju;
- Znak za razmak (bjelinu) nije dozvoljeno koristiti u imenu varijable (moguće je u tu svrhu iskoristiti znak podcrta odnosno "_").

Preporuke koje je dobro poštivati tijekom zadavanja imena varijabli jesu:

- Ime varijable je dobro da bude smisleno ("*varijabla1*" je loše ime dok je "*prviPribrojnik*" mnogo jasnije ime za varijablu);
- Prvo slovo imena neka bude malo;
- Ukoliko se ime varijable sastoji od više riječi, dobro je da svaka sljedeća riječ počinje velikim slovom bez bjeline (npr. "*rezultatZbrojaTemperatura*").

Kada se kreće u rješavanje nekog problema, odnosno pisanje programa, uputno je prvo razraditi ideju programskog rješenja. U tu svrhu preporučljivo je prvo definirati potrebne varijable. Za definiciju potrebnih varijabli određuje se ime varijable, tip podatka koji će biti spremljen u varijablu te kratak komentar značenja podatka spremljenog u varijablu. Kratak komentar je opcionalan, a služi programeru kao dokumentacija tijekom pisanja programa. Primjer popisa varijabli definiranih za pisanje programa dan je u tablici 3.1. Kako se popis varijabli definira prije same razrade ideje moguć je slučaj da nisu predviđene sve potrebne varijable. Iz tog razloga moguće je popis varijabli prilagođavati tijekom razrade ideje programskog rješenja ili tijekom pisanja programskog kôda. Stoga je uputno napisati uz svaku varijablu kratak komentar odnosno objašnjenje kako bi se olakšao proces proširenja popisa varijabli.

Ime varijable	Tip varijable	Značenje varijable
brojiloVozilaD507	cijeli broj bez predznaka	Brojilo vozila na državnoj cesti D507 između čvorova Valentinovo i Gubaševo na čvoru Krapinske Toplice
tempMotora	decimalan broj	Temperatura motora broda izražena u stupnjevima Celsiusa
zelenoSvjetli	logička vrijednost	Logička oznaka da svijetli zeleno svjetlo na semaforu
imeStudenta	niz znakova	Ime studenta koji je upisao predmet Računalstvo

Tablica 3.1: Primjer popisa varijabli za kreiranje idejnog programskog rješenja.

Vrste varijabli

Varijable se mogu podijeliti prema području vidljivosti na globalne i lokalne. Prema mogućnosti promjene njihove vrijednosti tijekom izvođenja programa moguće je definirati varijable i konstante kojima se vrijednost pridruži jednom prilikom deklaracije i nakon toga se više ne može mijenjati. Područje vidljivosti označava područje programskog odsječka unutar kojeg je moguće pristupiti varijabli. Kod globalnih varijabli osigurana je vidljivost u cijelom programskom kôdu tako da je moguće sve varijable definirati na jednom mjestu. Na prvi pogled to se čini kao dobar pristup, no u stvarnosti se složeniji programi odnosno aplikacije koje se danas koriste sastoje od više datoteka s programskim kôdom. Također se koristi podjela pojedinih jednostavnijih dijelova programskog kôda na funkcije i pomoćne metode koje se pozivaju iz nadređenih funkcija i metoda odnosno iz glavne funkcije ili metode. Tu glavnu funkciju ili metodu poziva operacijski sustav prilikom pokretanja aplikacije. Uz korištenje globalnih varijabli smanjuje se mogućnost ponovnog korištenja istog programskog kôda u drugim programima. Iz tog se razloga najčešće koriste lokalne varijable koje su vidljive samo u programskom odsječku gdje im se i pristupa. Izvan tog programskog odsječka lokalne varijable ne postoje. Na taj način se postiže i ušteda u memoriji kako se memorija za pojedine varijable zauzima i oslobađa prema potrebi aplikacije. Za označavanje programskih odsječaka različiti programski jezici imaju različite oznake. U slučaju programskog jezika C♯, koji će se koristiti u ovoj skripti, početak programskog odsječka označava se s "{", a završetak s "}".

Različito područje vidljivosti, odnosno programski odsječak valjanosti globalnih i lokalnih varijabli, potrebno je uzeti u obzir prilikom deklaracije varijable. Razlog je što se unutar istog područja vidljivosti ne smiju deklarirati dvije varijable istog imena. U slučaju preklapanja područja vidljivosti moguće je imati deklaraciju varijabli s istim imenom s time da imaju biti deklarirane u različitim područjima vidljivosti. Pri tome se pojavljuje slučaj da manji programski odsječci postoje unutar većeg. Ako se unutar većeg i u njemu uključenog manjeg programskog odsječka deklarira varijabla istog imena tada je u području manjeg programskog odsječka valjana varijabla koja je deklarirana u sklopu programskog kôda tog manjeg programskog odsječka. Izvan njega valjana je varijabla koja je deklarirana u sklopu programskog kôda tog većeg programskog odsječka. Takav način rješavanja preklapanja područja vidljivosti varijabli koristi se i u programskom jeziku C♯.

Kod obrade podataka postoji potreba korištenja različitih konstanti kao što je to Ludolfov broj odnosno π, prirodni broj e, porezna stopa, dozvoljeni postotni iznos prekoračenja brzine na dionici autoceste, dozvoljena masa nekog modela zrakoplova i sl. Zajedničko je kod svih tih vrijednosti da se ne mijenjaju ili u slučaju poreznih stopa mijenjaju rijetko. Za definiciju ovakvih vrijednosti

najpogodnije je iskoristiti tip varijable kod koje se vrijednost definira prilikom njezine deklaracije i nakon toga njezinu vrijednost više nije moguće mijenjati u programu. U tu se svrhu koriste konstante. Najčešće su osnovne konstante već definirane u sklopu programskog jezika pa se tako u programskom jeziku C♯ vrijednost broja π dohvaća atributom "*System.Math.PI*", a vrijednost prirodnog broja e atributom "*System.Math.E*".

Promjena vrijednosti varijable

Kako bi se promijenila vrijednost varijable koristi se operator pridruživanja. Moguće ga je poistovjetiti sa znakom jednakosti "=" poznatim iz matematike pomoću kojeg se radi izjednačavanje vrijednosti s lijeve i desne strane tog znaka. Napiše li se kao primjer izraz:

$$cijeliBroj = 5, \tag{3.1}$$

znači da smo izjednačili vrijednost varijable "*cijeliBroj*" s vrijednošću 5. Svugdje gdje se dalje u nekom izrazu koristi varijabla "*cijeliBroj*" zna se da je njena vrijednost 5. U slučaju da se isti izraz 3.1 zada kao naredba koju će računalo izvršiti, desit će se pridruživanje vrijednosti s desne strane znaka "=" memorijskoj lokaciji na koju pokazuje ime varijable navedeno s lijeve strane znaka "=". Fizički to znači da se binaran podatak, koji se npr. nalazi u akumulatoru mikroprocesora, prenosi u memorijsku lokaciju navedene varijable. Pri tome se prije spremanja nove vrijednosti postojeća vrijednost u memorijskoj lokaciji varijable briše. Iz tog razloga je za svaki podatak, koji se želi spremiti u memoriju, potrebno deklarirati zasebnu varijablu. Ovisno o programskom jeziku koji se koristi, za operator pridruživanja se koriste različiti simboli. U okviru predmeta Računalstvo obrađuju se postupci rješavanja zadataka primjenom pseudokôda, dijagrama toka (program Raptor) te programskog jezika C♯. U tablici 3.2 dan je pregled različitih simbola operatora pridruživanja. Izgled pojedinog simbola potrebno je strogo poštivati u svakom programskom jeziku jer u suprotnom napisani programski kôd neće proći provjeru ispravnosti. Često se kod dijagrama toka koristi ista oznaka operatora pridruživanja kao za programski jezik C♯ pa će to tako i u ovoj skripti. Prilikom implementacije dijagrama toka u program Raptor potrebno je onda napraviti prilagodbu simbola pojedinih operatora te naredbi aritmetičkih, logičkih i trigonometrijskih funkcija.

Isti princip promjene vrijednosti varijable koristi se prilikom različitih matematičkih operacija. Kao najjednostavniji primjer može se uzeti operacija zbrajanja. Za početak se može pretpostaviti da su u računalu deklarirane tri varijable ("*prviPribrojnik*", "*drugiPribrojnik*" i "*zbroj*") i da su im pridružene početne vrijednosti korištenjem izraza 3.1. Kako bi se odredio zbroj dva pribrojnika kao

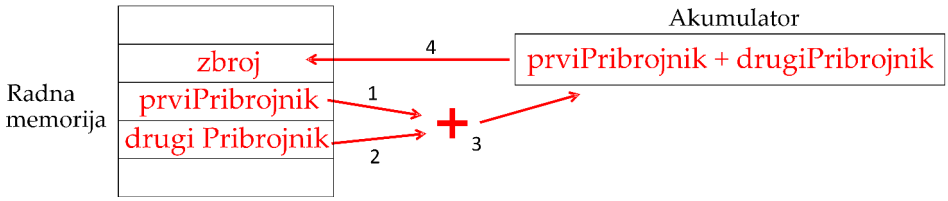
Matematički simbol	Pseudokôd	Dijagram toka (Raptor)	Programski jezik C♯
=	:=	←	=

Tablica 3.2: Pregled simbola operatora pridruživanja u okviru predmeta Računalstvo.

naredba pseudokôda može se napisati sljedeći izraz:

$$zbroj := prviPribrojnik + drugiPribrojnik. \tag{3.2}$$

Sve tri varijable smještene su u memoriji i prilikom izvršavanja ove naredbe u računalu se izvršavaju koraci kao što je prikazano na slici 3.2. Brojevima su označeni pojedini koraci i kao prvi korak se izvodi dohvat vrijednosti varijable "*prviPribrojnik*". Vrijednost se iz radne memorije unosi u jedan od registara akumulatora u mikroprocesoru pri čemu vrijednost varijable u radnoj memoriji ostaje sačuvana. Današnji mikroprocesori imaju više registara kojima se pristupa izravno ili se pristupa samo prvom registru. Kao drugi korak izvodi se dohvat vrijednosti varijable "*drugiPribrojnik*" i ona se također sprema u jedan od registara akumulatora u mikroprocesoru. Pri tome se postojeća vrijednost iz prvog registra prebacuje u sljedeći registar akumulatora dok se postojeća vrijednost zadnjeg registra akumulatora gubi. Tek kada su obje vrijednosti varijabli u registrima akumulatora izvršava se sama operacija zbrajanja kao treći korak. Rezultat operacije zbrajanja privremeno se sprema u prvi registar akumulatora i tek se u sklopu zadnjeg, četvrtog koraka dobiveni rezultat sprema u radnu memoriju računala. Taj zadnji, četvrti korak označava izvršavanje operatora pridruživanja.

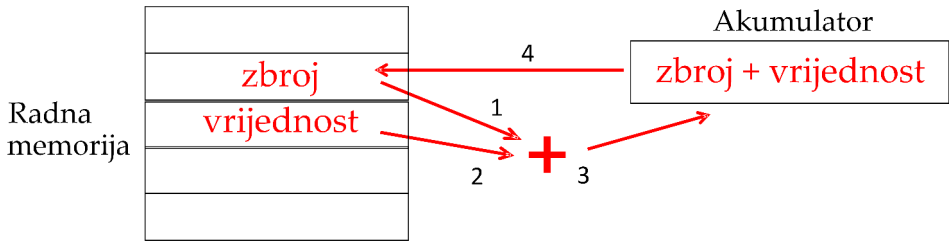


Slika 3.2: Prijenos podataka unutar računala prilikom zbrajanja.

Opisani koraci izvršavanja jednostavnije operacije zbrajanja omogućuju u računalu implementaciju brzih promjena vrijednosti varijable, odnosno inkrement (povećanje) i dekrement (smanjenje) vrijednosti varijable za 1. Tu je posebnost moguće izraziti sljedećom naredbom pseudokôda:

$$zbroj := zbroj + vrijednost. \tag{3.3}$$

Analizira li se izraz 3.3 po pravilima matematike može se odmah zaključiti kako ne vrijedi. Ako se izraz 3.3 analizira kao naredba koju će izvesti računalo, odnosno kao pridruživanje vrijednosti s desne strane varijabli koja se nalazi s lijeve strane znaka pridruživanja, moguće ju je razložiti na pojedine korake prikazane na slici 3.3. Kao prvi korak ponovno se izvodi dohvat vrijednosti prve varijable (u ovom slučaju varijable "*zbroj*") iz memorije u akumulator mikroprocesora. Kao drugi korak ponovno se izvodi dohvat vrijednosti druge varijable (u ovom slučaju varijable "*vrijednost*") iz memorije u akumulator mikroprocesora. Kao treći korak izvodi se operacija zbrajanja i rezultat zbrajanja ostaje u akumulatoru mikroprocesora. Ovdje je potrebno napomenuti da su vrijednosti varijabli "*zbroj*" i "*vrijednost*" u radnoj memoriji računala još uvijek nepromijenjene. U četvrtom koraku se vrijednost iz akumulatora odnosno rezultat zbrajanja i time nova vrijednost varijable "*zbroj*" sprema u radnu memoriju. Pri tome se početna vrijednost varijable "*zbroj*" gubi i u radnoj memoriji ostaje samo nova vrijednost dobivena inkrementom. Ovakav način uvećanja vrijednosti varijable je specifičan za način rada računala i moguće ga je primijeniti u svim programskim jezicima.



Slika 3.3: Prijenos podataka unutar računala prilikom inkrementa varijable.

Zamjena vrijednosti dvije varijable

Tijekom izvršavanja programa ponekad se pojavi potreba za zamjenom vrijednosti između dvije varijable. U tablici 3.3 prikazan je rezultat jedne takve zamjene vrijednosti između varijabli. Kako bi se ova operacija mogla ostvariti potrebno je imati na umu opisani način rada s varijablama. Intuitivno neispravno rješenje ovog problema dano je sljedećim dvjema operacijama:

$$prviOperand := drugiOperand, \tag{3.4}$$

$$drugiOperand := prviOperand. \tag{3.5}$$

Rezultat operacija dan izrazima 3.4 i 3.5 biti će takav da će vrijednosti obje varijable biti jednaka 3. Razlog toga je što izvršavanje operacije dane izrazom 3.4 u varijablu "*prviOperand*" upisuje vrijednost varijable "*drugiOperand*" (vrijednost 3 u razmatranom primjeru) dok se postojeća vrijednost varijable "*prviOperand*" briše i više nije spremljena u memoriji.

Ime varijable	Početna vrijednost	Konačna vrijednost
prviOperand	1	3
drugiOperand	3	1

Tablica 3.3: Rezultat zamjene vrijednosti između dvije varijable.

Ispravno rješenje ovog problema uključuje korištenje jedne dodatne pomoćne varijable. U nju je prvo potrebno spremiti početnu vrijednost varijable "*prviOperand*" kako bi ostala sačuvana nakon što joj se pridruži vrijednost varijable "*drugiOperand*". U zadnjem se koraku varijabli "*drugiOperand*" pridruži vrijednost varijable "*prviOperand*" sačuvana u pomoćnoj varijabli. Opisana procedura može se formalno opisati sljedećim izrazima:

$$pomocna := prviOperand, \tag{3.6}$$

$$prviOperand := drugiOperand, \tag{3.7}$$

$$drugiOperand := pomocna. \tag{3.8}$$

3.2 Tipovi podataka

U računalu se za prikaz negativnih i decimalnih brojeva ne može koristiti znak predznaka i decimalnog zareza ili točke, već se ta oznaka kôdira u pripadnom formatu za prikaz pojedine vrste numeričkog podatka [9]. U tu su svrhu definirani različiti tipovi podataka, odnosno načini kôdiranja numeričkih, logičkih i znakovnih podataka. Za svaki tip podatka koristi se binaran prikaz, a za interpretaciju podataka bitno je znati format zapisa podatka odnosno način kôdiranja. Interpretacija podataka se odvija automatski prilikom njihove obrade korištenjem značajki pojedinog tipa podatka. Definicija tipova podataka naročito je bitna kada se koriste viši programski jezici kod kojih nema automatizma u kreiranju

varijabli. Jedan od takvih jezika je i C♯. Prilikom kreiranja odnosno razrade ideje programskog rješenja zadanog problema, deklaracija nije toliko bitna koliko je bitno ispravno korištenje varijabli. Tako se u programskom alatu za kreiranje dijagrama toka Raptor, varijabla automatski deklarira prilikom njenog prvog korištenja, odnosno kada joj se prvi puta pridružuje vrijednost. Također se automatski podesi i ispravan tip podatka.

3.2.1 Prikaz cijelih brojeva

Za prikaz cijelih brojeva unutar računala koriste se zakonitosti opisane unutar dijela skripte vezanog za binarni brojevni sustav u odjeljku 2.1.2. Moguće je prikazati samo pozitivne cijele brojeve (bez predznaka) ili cijele brojeve s predznakom. U slučaju prikaza bez predznaka, najveća vrijednost je nešto veća što pokazuje usporedba izraza 2.3 i 2.20. Negativni se brojevi u računalu prikazuju pomoću dvojnog komplementa.

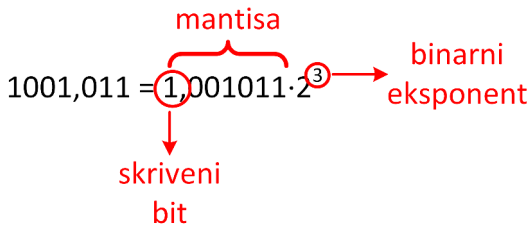
U sklopu programskog jezika C♯ cijeli se brojevi mogu prikazati pomoću tri različita tipa podatka. Razlikuju se prema količini memorije koja se koristi za prikaz. Ovisno o očekivanom rasponu vrijednosti podataka, koji se spremaju u pojedinu varijablu, može se odabrati tip koji će zauzeti najmanje memorije. Za prikaz cijelih brojeva bez predznaka pojedinoj ključnoj riječi dodaje se prefiks "*u*" koji označava englesku riječ "unsigned" odnosno "*bez predznaka*". U tablici 3.4 dan je pregled tipova podataka za prikaz cijelih brojeva u programskom jeziku C♯. Tablica sadrži ključnu riječ za deklaraciju varijable određenog raspona cijelih brojeva, količinu memorije potrebno za spremanje varijable te iznose najmanje i najveće vrijednosti. Količina memorije, koja se koristi za spremanje pojedinog tipa podatka, mijenja se s razvojem računala i povećavanjem potrebe za spremanje velikih količina podataka. Iz tog je razloga potrebno za detalje uvijek provjeriti trenutno važeću normu korištenog programskog jezika.

Ključna riječ	Količina memorije (bit/bajt)	Najmanja vrijednost	Najveća vrijednost
short	$16b$ / $2B$	-32.768	32.767
ushort	$16b$ / $2B$	0	65.535
int	$32b$ / $4B$	$-2.147.483.648$	2.147.483.647
uint	$32b$ / $4B$	0	4.294.967.295
long	$64b$ / $8B$	$-9.223.372.036.854.775.808$	9.223.372.036.854.775.807
ulong	$64b$ / $8B$	0	18.446.744.073.709.551.615

Tablica 3.4: Pregled tipova podataka za spremanje cijelih brojeva u programskog jeziku C♯.

3.2.2 Prikaz brojeva s plivajućim zarezom

Kod prikaza brojeva s plivajućim zarezom (engl. "floating point numbers") odnosno realnih brojeva koristi se bit za prikaz predznaka i tzv. normalizirani eksponencijalni zapis [13]. Ideja normaliziranog eksponencijalnog zapisa jest prikaz binarnog broja s jednom binarnom znamenkom ispred decimalnog zareza pomnoženog odgovarajućom potencijom broja 2. Tako se binarni broj može prikazati pomoću mantise i binarnog eksponenta. Kako je prva binarna znamenka mantise po definiciji prikaza uvijek jedinica, taj bit nije potrebno spremati pa se naziva i tzv. skriveni bit. Primjer jednog takvog skrivenog bita prikazan je na slici 3.4. Iz ovog primjera moguće je vidjeti kako ovakav prikaz omogućuje prikaz vrlo velikih i vrlo malih binarnih brojeva bez korištenja velikog broja nula.



Slika 3.4: Primjer normaliziranog binarnog broja.

Normalizirani prikaz je ujedno i osnova za prikaz brojeva s plivajućim zarezom pomoću norme $IEEE754$ [9]. Sastavni dijelovi za prikaz u memoriji računala su bit predznaka (P), karakteristika (K) te mantisa bez skrivenog bita (M). Ovisno o količini memorije iskorištene za prikaz broja moguće je dobiti različite preciznosti, odnosno broj točno prikazanih decimala. Često se koriste $32b$ za prikaz što se naziva prikaz u jednostrukoj preciznosti. Princip prikaza decimalnog broja korištenjem norme $IEEE754$ u jednostrukoj preciznosti dan je na slici 3.5. Za prikaz predznaka koristi se jedan bit, za prikaz karakteristike 8 bita, a za prikaz mantise bez skrivenog bita 23 bita.

Slika 3.5: Prikaz decimalnog broja u jednostrukoj preciznosti pomoću $IEEE754$ norme.

Kod prikaza predznaka P koristi se isti princip kao i kod dvojnog komple-

menta. Vrijednost 0 označava pozitivan broj, a vrijednost 1 negativan broj. Karakteristika K služi za prikaz binarnog eksponenta (BE) koji može biti pozitivan ili negativan. Kako je karakteristika uvijek pozitivna u području $[0,255]$ koristi se preslikavanje dozvoljenog raspona binarnog eksponenta ($BE \in [-126,127]$) u dozvoljeno područje karakteristike korištenjem sljedećeg izraza:

$$K = BE + 127. \tag{3.9}$$

Promotri li se pobliže dozvoljeni raspon karakteristike i njen izračun pomoću izraza 3.9 moguće je zaključiti kako se ne koriste sve vrijednosti karakteristike. Točnije vrijednosti 0 i 255 nisu iskorištene. One su rezervirane za sljedeće specijalne slučajeve vrijednosti decimalnog broja X:

- **Ništica** uz $K = 0$ i $F = 0 \Longrightarrow X = 0$;
- **Preljev** uz $K = 255$ i $F = 0 \Longrightarrow BE > 127$;
- **Podljev** uz $K = 0$ i $F \neq 0 \Longrightarrow BE < -126$;
- **Neispravan broj** (NAN, engl. "Not a Number") uz $K = 255$ *i* $F \neq 0$.

Specifičan slučaj preljev označava da se broj ne može prikazati jer je prevelik, podljev označava da se broj ne može prikazati jer je premalen, a neispravan broj označava neispravan rezultat neke operacije (npr. dijeljenje s nulom). Ispravno pretvoren broj X s plivajućim zarezom u ovom se prikazu može izraziti i pomoću sljedećeg izraza:

$$X = (-1)^P \cdot 1,F \cdot 2^{BE}. \tag{3.10}$$

U programskom jeziku C♯ postoji nekoliko tipova za prikaz brojeva s plivajućim zarezom. Razlikuju se po količini memorije potrebnoj za prikaz. Njihov pregled je dan u tablici 3.5. Moguće je primijetiti da tip "*decimal*" za prikaz koristi najviše memorije, a može prikazati najmanji raspon. Radi se o tome da taj tip ima najveću preciznost, odnosno može točno prikazati veći broj decimala od ostala dva tipa. To je postignuto tako da je količina bitova namijenjena karakteristici smanjena radi povećanja broja bitova namijenjenih mantisi bez skrivenog bita.

3.2.3 Prikaz logičkih vrijednosti

Logičke vrijednosti vezane su za Boole-ovu algebru kao dio matematičke logike. Unutar nje definirane su logičke funkcije koje obrađuju logičke varijable. Posebnost logičkih varijabli je da mogu imati samo dvije moguće vrijednosti: istina

Ključna riječ	Količina memorije (bit/bajt)	Najmanja vrijednost	Najveća vrijednost
float	$32b$ / $4B$	$1,5 \cdot 10^{-45}$	$3,4 \cdot 10^{38}$
double	$64b$ / $8B$	$5,0 \cdot 10^{-324}$	$1,7 \cdot 10^{308}$
decimal	$128b$ / $16B$	$1,0 \cdot 10^{-28}$	$7,9 \cdot 10^{28}$

Tablica 3.5: Pregled tipova podataka za spremanje brojeva s plivajućim zarezom u programskog jeziku C♯.

(engl. "true", logička jedinica) i neistina ili laž (engl. "false", logička nula). Radi jednostavnosti obrade podataka definirano je preslikavanje numeričke vrijednosti u logičku vrijednost. U tom preslikavanju vrijedi da numerička vrijednost 0 odgovara logičkoj vrijednosti laž, a svaka druga numerička vrijednost odgovara logičkoj vrijednosti istina. Detalji o logičkim funkcijama bit će objašnjeni u sljedećem poglavlju.

U sklopu programskog jezika C♯ postoji poseban tip podataka za prikaz logičkih vrijednosti. Ključna riječ za deklaraciju logičkih varijabli je "*bool*". Deklarirana logička varijabla može poprimiti dvije različite vrijednosti: za logičku vrijednost istine je to "*true*", a za logičku vrijednost laži je to "*false*". Kod zadavanja imena varijable vrijede prethodno navedena pravila.

3.2.4 Prikaz znakova

Osim numeričkih i logičkih vrijednosti u računalu je potrebno prikazati i ostale vrste podataka. Jedna od takvih vrsta podataka su znakovi čiji nizovi čine tekst. Tekstualni podaci se u računalu također prikazuju pomoću binarnog brojevnog sustava pri čemu se koristi kôdiranje u odgovarajućem tipu podatka. Svakom znaku pridružena je određena numerička cjelobrojna vrijednost.

Za prikaz znakova najčešće se koriste široko raširenі kôdovi ASCII (Američki standardni kôd za razmjenu informacija, engl. "American Standard Code for Information Interchange"), a od 1991. godine i UNICODE. Prvo je nastao ASCII kôd i prikazivao je znakove unutar $7b$ [9]. Kako je memorija računala organizirana u bajtove, najviši bit znaka unutar ASCII kôda ima vrijednost nula da se dobije punih $8b$. ASCII kôd omogućuje prikaz malih i velikih slova engleske abecede, dekadskih znamenaka, interpunkcijskih znakova te specijalnih/upravljačkih znakova. Za prikaz svake skupine znakova unutar ASCII kôda je rezervirano određeno numeričko područje kao što je prikazano u tablici 3.6. Specijalni/upravljački znakovi služe uglavnom za upravljanje ispisom, a kao primjer se mogu navesti znakovi: DEL (engl. "delete", vrijednost 127) za brisanje

prvog znaka ispred kursora, CR (engl. "carriage return", vrijednost 13) za pomak kursora na početak retka, LF (engl. "line feed", vrijednost 10) za pomak kursora jedan redak naniže i dr. ASCII kôd se obično izdaje u obliku tablice koja sadrži popis numeričkih vrijednosti, simbol pripadnog znaka te komentar.

Skupina znakova	Numerička vrijednost (kôd)
Mala slova engleske abecede ($a \ldots z$)	$97 \ldots 122$
Velika slova engleske abecede ($A \ldots Z$)	$65 \ldots 90$
Dekadske znamenke $0 \ldots 9$	$48 \ldots 57$
Interpunkcijski znakovi	$32 \ldots 47$, $58 \ldots 64$ $91 \ldots 96$, $123 \ldots 126$
Specijalni/upravljački znakovi	$0 \ldots 31$ i 127

Tablica 3.6: Pregled numeričkih vrijednosti kôda ASCII za pojedini skup znakova.

Nedostatak ASCII kôda je što može prikazati samo 127 znakova. Ta količina znakova nije dovoljna za prikaz posebnih znakova u svim svjetskim jezicima. Prijelazno rješenje je bilo kreiranje lokalnih izvedenica ASCII kôda koji uključuje prikaz posebnih znakova pojedinog jezika. Za primjer se može navesti CROSCII koji je uključio prikaz hrvatskih znakova: č, š, ć, đ i ž. Danas se za kôdiranje znakova u računalu koristi UNICODE koji omogućuje prikaz svih znakova svjetskih jezika. Postoji nekoliko inačica kôda UNICODE koji se međusobno razlikuju po količini bajtova koji se koriste za prikaz jednog znaka. Tako se često koristi inačica koja po jednom znaku koristi $2B$ odnosno $4B$. UNICODE je nastao na osnovi 8 bitnog ASCII kôda i za znakove koji su definirani u ASCII kôdu UNICODE koristi iste numeričke vrijednosti, odnosno isti kôd.

Za prikaz znakova su u programskom jeziku C♯ dostupna dva tipa podatka, za prikaz jednog znaka "*char*" i za prikaz niza znakova "*string*". Za prikaz jednog znaka pomoću tipa "*char*" koriste se $2B$ i znak se prikazuje pomoću kôda UNICODE. Za prikaz vrijednosti tipa "*char*" koriste se jednostruki navodnici pa se tako varijabla tipa "*char*" može deklarirati i inicijalizirati pomoću linije kôda "*char jedanZnak = 'a';*". Prikaz samo jednog znaka nije dovoljan za spremanje tekstova pa se analogno koristi tip "*string*" s mogućnošću prikaza neograničeno dugog niza znakova. Jedino ograničenje predstavlja memorija dostupna računalu. Za prikaz vrijednosti tipa "*string*" koriste se dvostruki navodnici pa se varijabla tipa "*string*" može deklarirati i inicijalizirati pomoću linije kôda "*string nizZnakova = "Vrijednost varijable nizZnakova!";*". Za prikaz jednog znaka unutar niza znakova koriste se $2B$ i potrebna količina memorije se zauzima automatski. Pri tome deklarirana varijabla tipa "*string*" postaje u programskom jeziku C♯ objekt. U računalstvu objekt je svaka memorijska lokacija koja sadrži neku vrijednosti i može biti referencirana nekim identifikatorom. Objekt tako može

biti varijabla, podatkovna struktura, funkcija ili metoda. U slučaju varijable u memoriji je spremljena njena vrijednost koja se referencira imenom varijable uz mogućnost dohvata dodatnih informacija o varijabli korištenjem pomoćnih metoda i atributa. Tako se duljina znakovnog niza može jednostavno dohvatiti atributom "*Length*" odnosno za dani primjer naredbom "*nizZnakova.Length*". Može se primijetiti kako se koristi operator "." za pristup atributu pojedinog objekta.

3.3 Rad s varijablama u programskom jeziku C♯

U višim programskim jezicima kao što je to i programski jezik C♯ potrebno je na početku programa uvijek napraviti deklaraciju varijabli. Ona uključuje popis svih varijabli, koje će se koristiti u programu, i njihovih tipova. Tek nakon deklaracije se varijabla smije koristiti u programu. Korak deklaracije varijabli uključuje definiranje tipa varijable te imena varijable. Prilikom izvođenja programa računalo, odnosno operacijski sustav računala prilikom izvršavanja naredbe deklaracije varijable traži slobodno memorijsko područje u radnoj memoriji, zauzima ga za pripadnu varijablu i pridružuje adresu prvog bajta u nizu simboličkom imenu varijable. Tako je računalu omogućeno da dohvati vrijednost varijable, a programeru olakšano pisanje programskog kôda.

Kod deklaracije varijable postoji uvijek pitanje početne vrijednosti varijable. Danas se u većini slučajeva prilikom deklaracije varijabli pridružuje početna vrijednost 0, odnosno prazan niz znakova. To prije nije bio slučaj, već su kao početna vrijednost bile zaostale vrijednosti u memoriji od programa koji su završili sa svojim izvršavanjem. Iz tog razloga se preporučuje uvijek nakon deklaracije varijable zadati joj početnu vrijednost pomoću operatora pridruživanja, odnosno inicijalizirati varijablu. Ovisno o programskom jeziku koriste se različiti simboli operatora pridruživanja kao što je to prikazano u tablici 3.2. Naravno, kod pridruživanja početne vrijednosti potrebno je obratiti pažnju na ispravnu vrijednost prema deklariranom tipu varijable te na ispravan zapis prema korištenom programskom jeziku.

Za deklaraciju cjelobrojne varijable u programskom jeziku C♯ dostupni su tipovi dani u tablici 3.4. Primjeri deklaracije cjelobrojnih varijabli dani su programskim kôdom 3.1. Komentarima (počinju oznakom "//") zelene boje označeni su i objašnjeni pojedini dijelovi programskog kôda. Tako se prvi dio prorgamskog kôda odnosi na deklaraciju, a drugi dio na pridruživanje početne vrijednosti, odnosno inicijalizaciju. Moguće je primijetiti da svaka linija kôda završava znakom ";" koji u programskom jeziku C♯ označava kraj jedne naredbe. Dodatno, u sklopu jedne naredbe moguće je deklarirati više varijabli istog tipa. Također je moguće zadati istu početnu vrijednost više varijabli. Isti princip

vrijedi za deklaraciju i inicijalizaciju ostalih tipova varijabli.

Programski kôd 3.1: Primjer deklaracije i inicijalizacije cjelobrojnih varijabli.

```
// deklaracija varijabli
short maliCijeli;
int cijeliBroj;
uint pozitivanCijeli;
long prviCijeli, drugiCijeli, treciCijeli;

// inicijalizacija varijabli
maliCijeli = 0;
cijeliBroj = -10;
pozitivanCijeli = 10;
prviCijeli = drugiCijeli = treciCijeli = 1000000000;
```

3.4 Zadaci za samostalan rad

■ Zadatak 3.1

Programer izrađuje program za dijeljenje dva cijela broja. Potrebno je napraviti tablicu potrebnih varijabli za program. Objasnite odabir tipa pojedine varijable!

■ Zadatak 3.2

Predviđeni najveći dnevni broj vozila na dionici autoceste iznosi 65.000 *vozila. Koji se sve tipovi varijabli mogu koristiti za prikaz broja vozila? Objasnite odabir!*

■ Zadatak 3.3

Napisani program služi za množenje dva cijela broja. Može li varijabla u koju će se spremiti umnožak biti istog tipa kao i množitelji? Objasnite odgovor!

■ Zadatak 3.4

Programer izrađuje program za simulaciju leta zrakoplova i nastoji postići što veću točnost simulacije. Koji tip varijable bi bilo uputno da programer koristi? Objasnite odabir!

Poglavlje 4

Logičke funkcije

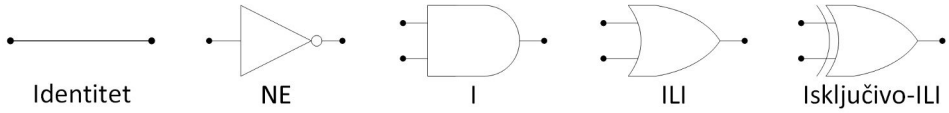
Logičke su funkcije temelj na kojem počiva izvođenje svih računskih i logičkih operacija (zbrajanja, oduzimanja, množenja, dijeljenja i grananja) na računalima čiji rad se zasniva na dva stanja. Često ta stanja nazivamo "istina" i "laž", a možemo ih označavati kao "uključeno" i "isključeno" ili ih simbolički opisivati s "1" i "0". U ovom poglavlju objasnit će se najčešće korištene logičke funkcije: identitet, negaciju, I, (uključivo) ILI te isključivo-ILI.

4.1 Osnove Booleove algebre

Booleova algebra ili logička algebra služi za jednostavnije opisivanje skupova elemenata i njihovih odnosa. Digitalna računala kakvima se danas koristimo zasnivaju svoj rad na dva stanja pa Booleova algebra omogućava vrlo dobru primjenu u razumijevanju operacija koje računala izvode. Konstrukcija računala provodi se od jednostavnih sklopova ka složenijima i oslanja se na analizi rada tih sklopova spojenih u složene sustave. S obzirom na to da računala mogu imati samo dva stanja, "0" i "1", za analizu njihova rada pogodan je binarni sustav, koji je objašnjen u potpoglavlju 2.1.2. Booleova algebra preko logičkih funkcija definira jednostavna pravila kojima se utvrđuju ishodi dvaju događaja (operanada ili varijabli) s obzirom na njihova početna stanja. Za prikaz svih mogućih stanja varijabli i njihovih ishoda pod djelovanjem neke logičke funkcije služe tablice istinitosti. Tablice istinitosti praktičan su alat za zapisivanje ishoda u slučaju manjeg broja varijabli, ali postaju prevelike u slučaju većeg broja događaja jer im broj redaka raste s potencijom broja 2 ($2^{broj\ varijabli}$).

4.2 Logičke funkcije

Logičke funkcije i njihovo djelovanje koje će se obraditi u ovom poglavlju može se objasniti na više načina pa će se tako osim tablice istinitosti koristiti logički sklopovi, koji se još nazivaju i logičkim vratima te analogiju sa strujnim krugom i prometnim problemima. Uporaba analogija ima za cilj pojednostavniti shvaćanje pojedine logičke funkcije na stvarnom primjeru.



Slika 4.1: Prikaz oblika logičkih sklopova kojima shematski prikazujemo pojedine logičke funkcije.

4.2.1 Logička funkcija identiteta

Logička funkcija identiteta najjednostavnija je logička funkcija kod koje početno stanje istinitosti nekog događaja, odnosno varijable, uvjetuje ishod identičan tome stanju. Ta funkcija primjenjuje se na svaku varijablu zasebno pa ju smatramo logičkom funkcijom jedne varijable. Logičku funkciju identiteta možemo prikazati tablicom istinitosti 4.1.

A	Z (ishod) = A
1	1
0	0

Tablica 4.1: Tablica istinitosti logičke funkcije identiteta.

Prometna analogija: Ako je na semaforu zeleno svjetlo, nastavit ću vožnju ravno. Ako na semaforu nije zeleno svjetlo, neću nastaviti vožnju ravno.

4.2.2 Logička funkcija NE

Logička funkcija negacije ili logički "NE" početno stanje istinitosti zamjenjuje s njemu suprotnim, komplementarnim stanjem. Ovo je također funkcija jedne varijable. Lako je pretpostaviti kako NE ima invertiranu tablicu ishoda u odnosu na funkciju identiteta pa se ova funkcija često naziva i inverterom.

Prometna analogija: Ako prijeđem cestu, neću ostati na istoj strani. Ako ne prijeđem cestu, neću doći na drugu stranu.

A	Z (ishod) = -A
1	0
0	1

Tablica 4.2: Tablica istinitosti logičke funkcije NE.

4.2.3 Logička funkcija I

Logička funkcija I ubraja se u funkcije koje djeluju na dvije varijable. Kako je broj varijabli (događaja) dvostruko veći nego kod prethodne dvije funkcije, tako će i tablica istinitosti 4.3 biti dvostruko veća, tj. imati će dvostruko više redaka, nego u slučaju jedne varijable. Logičku funkciju I možemo prikazati i pomoću analogije sa strujnim krugom, slika 4.2, pri čemu nam serijske sklopke prikazuju varijable A i B, a njihov rezultat žaruljica Z koja svijetli ili ne svijetli, ovisno o položaju sklopki. Možemo reći kako je nužan uvjet pri kojem će žaruljica svijetliti taj da obje sklopke budu u zatvorenom položaju, jer će jedino tako strujnim krugom teći struja.

A	B	Z = A • B
1	1	1
1	0	0
0	1	0
0	0	0

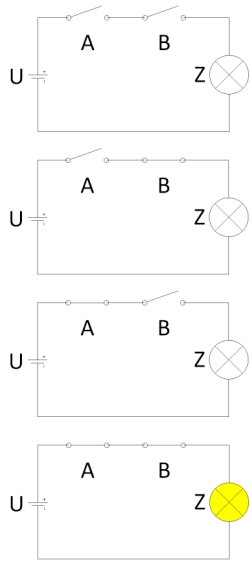
Tablica 4.3: Tablica istinitosti logičke funkcije I.

Prometna analogija: Ako je na pješačkom semaforu zeleno i automobili su zaustavljeni, prijeći ću cestu.

4.2.4 Logička funkcija (uključivo)-ILI

Logička funkcija ILI daje istinit ili pozitivan rezultat ako je barem jedan od uvjeta ispunjen, tj. ako barem jedna od varijabli ima vrijednost 1, kao što je prikazano u tablici 4.3. Logičku funkciju ILI također se može prikazati pomoću analogije sa strujnim krugom, slika 4.3, pri čemu položaj paralelnih sklopki tvori uvjete A i B, a rezultat logičke operacije prikazuje žaruljica Z koja svijetli ili ne svijetli, ovisno o položaju sklopki. U ovom slučaju strujni krug je zatvoren ako je samo jedna od sklopki u zatvorenom položaju ili ako su obje u zatvorenom položaju pa će u tri od četiri moguća slučaja žaruljica svijetliti.

Prometna analogija: Na predavanje mogu stići ako idem pješke ili ako idem biciklom.



Slika 4.2: Prikaz logičke funkcije I pomoću sklopki i žaruljice.

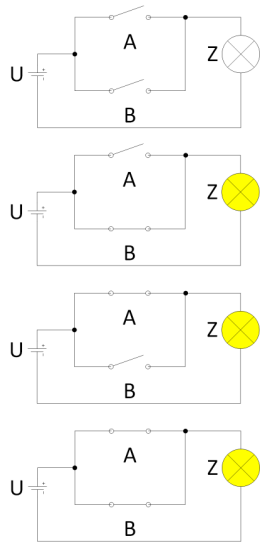
A	B	Z = A + B
1	1	1
1	0	1
0	1	1
0	0	0

Tablica 4.4: Tablica istinitosti logičke funkcije ILI.

4.2.5 Logička funkcija isključivo-ILI

Logička funkcija isključivo-ILI daje istinit ili pozitivan rezultat samo ako je jedan od uvjeta ispunjen, tj. ako samo jedna od ulaznih varijabli ima vrijednost 1, kao što je prikazano u tablici 4.5. Zanimljivo je napomenuti da se logička funkcija isključivo-ILI može ostvariti uporabom tri osnovne logičke funkcije: negacije, I, ILI.

Prometna analogija: Vlak u podne može biti u Zagrebu (isključivo) ili na putu prema Zagrebu.



Slika 4.3: Prikaz logičke funkcije ILI pomoću sklopki i žaruljice.

A	B	Z = A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

Tablica 4.5: Tablica istinitosti logičke funkcije isključivo-ILI.

4.3 Teoremi Booleove algebre

Teoremi Booleove algebre služe za smanjenje broja članova, odnosno složenosti logičke funkcije u situacijama kada imamo velik broj varijabli i operacija koje se provode nad njima. Stoga je važno usvojiti neke osnovne ekvivalencije koje pomažu pri pojednostavljenju zadanih formula. U tablici 4.6 dan je popis ekvivalencija koje se mogu koristiti pri pojednostavljenju logičkih izraza.

4.4 Primjeri

U nastavku su dani primjeri vezani za izradu logičkih izraza, njihovo pojednostavljenje, realizaciju logičke funkcije pomoću logičkih sklopova te određivanje tablica istinitosti. Kao dodatni primjeri se mogu proučiti i primjeri kreiranja

Komutativnost	A + B = B + A	A • B = B • A
Asocijativnost	(A + B) + C = = A + (B + C)	(A • B) • C = = A • (B • C)
Distributivnost	A • (B + C) = = (A • B) + (A • C)	A + (B • C) = = (A + B) • (A + C)
Neutralni element	A + 0 = A A + A = A	A • 1 = A A • A = A
Komplementarnost	A + (-A) = 1	A • (-A) = 0
De Morganovi zakoni	-(A + B) =(-A) • (-B)	-(A • B) =(-A) + (-B)
Involutivnost	-(-A) =A	
Anihilacija	A + 1 = 1	A • 0 = 0
Apsorpcija	A • (A + B) = A	A + A • B = A

Tablica 4.6: Teoremi Booleove algebre.

uvjeta za grananja i petlje dani u sljedećim poglavljima.

■ **Primjer 4.1**

Potrebno je reducirati izraz Z = (A + B) • (-(A • B)) na što manji broj varijabli i operatora potrebnih za realizaciju.

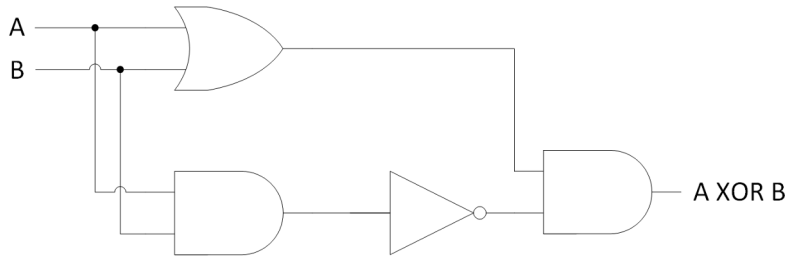
Rješenje:

Kao prvi korak napisat ćemo tablicu istinitosti. Funkcija se sastoji od dva operanda pa će tablica ukupno imati 4 retka u koje pišemo sve moguće ishode.

A	B	A + B	-(A • B)	(A + B) • (-(A • B))
1	1	1	0	0
1	0	1	1	1
0	1	1	1	1
0	0	0	1	0

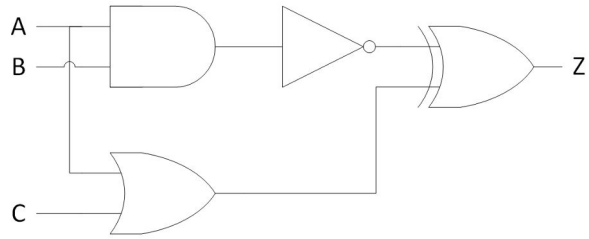
U zadnjem stupcu nalazi se konačno rješenje za koje možemo vidjeti da odgovara upravo logičkoj funkciji isključivo-ILI nad dva operanda. Iz dobivenog rezultata možemo zaključiti kako se funkcija isključivo-ILI ne ubraja u osnovne logičke funkcije već ju je moguće izvesti pomoću funkcija I, ILI i negacije. Ovaj primjer možemo prikazati i pomoću logičkih sklopova što je prikazano na slici 4.4.

■ **Primjer 4.2**



Slika 4.4: Realizacija logičke funkcije isključivo-ILI pomoću logičkih sklopova.

Napisati logički izraz i tablicu istinitosti za logički sklop prikazan slikom 4.5.



Slika 4.5: Logički sklop složenije logičke funkcije.

Rješenje:

Logički izraz koji opisuje ponašanje sklopa sa slike je sljedeći: Z = -(A • B) XOR (A + C).

A	B	C	-(A • B)	A + C	-(A • B) XOR (A + C)
1	1	1	0	1	1
1	1	0	0	1	1
1	0	1	1	1	0
1	0	0	1	1	0
0	1	1	1	1	0
0	1	0	1	0	1
0	0	1	1	1	0
0	0	0	1	0	1

4.5 Zadaci za samostalan rad

■ **Zadatak 4.1**

Nacrtajte logički sklop kojim realiziramo sljedeći logički izraz: Z = -((A • B) • C) XOR B.

■ **Zadatak 4.2**

Nacrtajte logički sklop kojim realiziramo sljedeći logički izraz: Z = (A • B) XOR C.

■ **Zadatak 4.3**

Nacrtajte logički sklop kojim realiziramo sljedeći logički izraz: Z = A + B • C i napišite tablicu istinitosti pazeći na redoslijed izvođenja operacija.

■ **Zadatak 4.4**

Primjenom teorema Booleove algebre potrebno je pojednostavniti logički izraz Z = (A • -(A + B)) • (B • (-C) +C).

■ **Zadatak 4.5**

Primjenom teorema Booleove algebre potrebno je pojednostavniti logički izraz Z = -(A) • -(B) + -(A) • B + A • B.

■ **Zadatak 4.6**

Primjenom teorema Booleove algebre potrebno je pojednostavniti logički izraz Z = -((A + B) • (A + -(B)) + ((-A) • B + A) • (-A).

Poglavlje 5

Matematičke operacije

U osnovne matematičke operacije ubrajaju se zbrajanje, oduzimanje, množenje i dijeljenje. Ovisno o izboru programskog jezika ili tipa podatka koji koristimo u nekom programskom jeziku, pojedine matematičke operacije izvodit će se na različite načine i s različitim rezultatom. Matematička operacija kod koje će se posebna pažnja usmjeriti na rezultat u ovisnosti o programskom jeziku i tipu podataka je operacija dijeljenja. Osim osnovnih matematičkim operacija, u ovom poglavlju pažnja će se posvetiti relacijskim i logičkim operacijama te njihovoj implementaciji u Raptoru (program za izradu dijagrama toka) i programskom jeziku C♯.

5.1 Aritmetički operatori

Aritmetički operatori definiraju koja aritmetička operacija se provodi nad ulaznim operandima. U standardnom zapisu aritmetičke operatore dijelimo na operatore zbrajanja (+), oduzimanja (−), množenja (·) ili (∗) i dijeljenja (/). U različitim programskim jezicima koristi se drugačiji oblik zapisa za istu aritmetičku operaciju. U tablici 5.1 navedeni su oblici zapisa za neke često korištene matematičke operatore u Raptoru i programskom jeziku C♯.

5.1.1 Različite vrste dijeljenja brojeva

U tablici 5.1 vidljivo je kako prve tri navedene matematičke operacije daju rezultat u onom tipu podatka u kojem su definirani operandi. Dijeljenje je pak matematička operacija koja se izvodi ponešto složenije jer njen rezultat često

Operator	Raptor	C♯
Zbrajanje	broj1 + broj2	broj1 + broj2
Oduzimanje	broj1 - broj2	broj1 - broj2
Množenje	broj1 * broj2	broj1 * broj2
Dijeljenje	broj1 / broj2	broj1 / broj2 (tip podatka float, double, decimal)
Cjelobrojno dijeljenje	floor(broj1 / broj2)	broj1 / broj2 (tip podatka short, int, long)
Ostatak cjelobrojnog dijeljenja	broj1 REM broj2 broj1 MOD broj2	broj1 % broj2
Potencija	broj1 ^ broj2 broj1 ** broj2	Math.Pow(broj1, broj2)
Drugi korijen	SQRT(broj)	Math.Sqrt(broj)
Apsolutna vrijednost	abs(broj)	Math.Abs(broj)
Prirodni logaritam	log(broj)	Math.Log(broj)
Najmanja vrijednost	min(broj1, broj2)	Math.Min(broj1, broj2)
Najveća vrijednost	max(broj1, broj2)	Math.Max(broj1, broj2)

Tablica 5.1: Oblici zapisa često korištenih matematičkih operatora u Raptoru i programskom jeziku C♯.

nije cjelobrojan pa je potrebno pažnju obratiti na točnost izvođenja operacije za pojedinu primjenu.

Ako želimo podijeliti dva cijela broja tipa podatka short, int ili long u programskom jeziku C♯, rezultat operacije cjelobrojnog dijeljenja bit će vrijednost zaokružena nadolje (engl. "floor"), odnosno cjelobrojni dio rezultata. Primjer za to je dijeljenje broja 3 s brojem 2, pri čemu je stvarni rezultat jednak $1,5$, a u slučaju cjelobrojnog dijeljenja rezultat će biti 1.

Ako nas cjelobrojni rezultat cjelobrojnog dijeljenja dva broja ne zadovoljava svojom točnošću, možemo posegnuti za drugim tipom podataka u programskom jeziku C♯, a to mogu biti float, double ili decimal. U tom slučaju rezultat primjera dijeljenja broja 3 s brojem 2 je $1,5$.

Osim cjelobrojnog dijeljenja i dijeljenja decimalnih brojeva, u nekim slučajevima potreban je i ostatak cjelobrojnog dijeljenja. Ostatak cjelobrojnog dijeljenja uvijek je broj manji od djelitelja, a rezultat dijeljenja cijelih brojeva može se dobiti kao kombinacija cjelobrojnog dijeljenja i ostatka cjelobrojnog dijeljenja. Cjelobrojnim dijeljenjem s ostatkom broja 8 s brojem 3 dobivamo rezultat od 2 i ostatak 2, jer je $2 \cdot 3 = 6$, a $6 + 2 = 8$. Implementacija u programskom jeziku

C♯ bi bila:

```
int a;
int b;
a = 8 / 3;
b = 8 % 3;
```

pri čemu se u varijablu "a" sprema rezultat cjelobrojnog dijeljenja, a u varijablu "b" ostatak cjelobrojnog dijeljenja.

5.1.2 Promjena vrijednosti brojača u petljama

Jedan od najčešćih pristupa kod izvođenja neke operacije ili skupa operacija nad više različitih elemenata je uporabom petlji. Petlje omogućavaju upravo takvo ponavljanje skupa naredbi nad nekim elementima pri čemu se mijenja indeks elementa kojem pristupamo. Kako se pri svakoj iteraciji petlje indeks elementa kojem se pristupa mijenja za 1, inkrementiranje u programskom jeziku C♯ izvodi se uporabom sintakse "$i++$" kojom povećavamo varijablu brojača "i" za jediničnu vrijednost. Osim povećavanja vrijednosti brojača, u nekim slučajevima je potrebno smanjivati vrijednost brojača, odnosno ostvariti kretanje kroz polje elemenata unazad. To se postiže uporabom izraza oblika "$i--$" kojim smanjujemo varijablu brojača "i" za jediničnu vrijednost.

Vrijednost brojača može se mijenjati i za iznos različit od jedinice. Pritom se u programskom jeziku C♯ koristi izraz oblika "$i+=velicinaInkrementa$", gdje je "*velicinaInkrementa*" broj za koji se uvećava brojač u svakom koraku petlje. Ako se prisjetimo izraza 3.3, za koji smo rekli kako nije striktno matematički korektan, nego pokazuje pridruživanje nove vrijednosti varijabli koja biva uvećana za neku vrijednost, možemo vidjeti kako se u slučaju brojača radi o potpuno istoj stvari. Veličina inkrementa je upravo "vrijednost" koja se dodaje broju zapisanom u varijabli "i" u trenutnom koraku petlje. Izraze koje smo dosad obradili pišemo vrlo slično u pseudokôdu, Raptoru i programskom jeziku C♯, kako je prikazano u tablici 5.2.

5.2 Logaritamske funkcije

Logaritamske funkcije često koristimo pri dobivanju numeričkih rezultata kod mnogih proračuna temeljenih na egzaktnim i empirijskim formulama. U praksi većinom koristimo logaritamske funkcije kojima je baza jednaka 10 ili pak bazi prirodnog logaritma ($e = 2,718$). Primjer ostvarenja logaritamskih funkcija u pseudokôdu, Raptoru i programskom jeziku C♯ prikazan je u tablici 5.3.

Opis	Pseudokôd	Raptor	C♯
Zbrajanje	+	+	+
Oduzimanje	−	−	−
Množenje	*	*	*
Dijeljenje	/	/	/
Cjelobrojno dijeljenje	div	div	/
Ostatak cjelobrojnog dijeljenja	mod	MOD, REM	%

Tablica 5.2: Primjeri načina pisanja pojedinih operatora u pseudokôdu, Raptoru i programskom jeziku C♯.

Logaritamska funkcija	Pseudokôd	Raptor	C♯
$log_{(10)}(a)$	Log(a)	Log(a)	Math.Log10(a)
$log_{(2)}(a)$	Log(a)/Log(2)	Log(a)/Log(2)	Math.Log(a)/Math.Log(2)
$ln(a)$	Ln(a)	Ln(a)	Math.Log(x)

Tablica 5.3: Zapisi logaritamskih funkcija u pseudokôdu, Raptoru i programskom jeziku C♯.

5.3 Trigonometrijske funkcije

U trigonometrijske funkcije ubrajaju se funkcije sinus, kosinus, tangens, kotangens, sekans i kosekans. U tablici 5.4 definirana su značenja pojedinih trigonometrijskih funkcija. Također su dani izrazi kako se vrijednost pojedine trigonometrijske funkcije može izračunati korištenjem stranica pravokutnog trokuta. Zapis trigonometrijskih funkcija u pseudokôdu, Raptoru i programskom jeziku C♯ prikazan je u tablici 5.5. Isti zapis koristiti će se u primjerima u narednim poglavljima. Bitno je imati na umu da u svim programskim jezicima ulazni argument (vrijednost) za izračun rezultata trigonometrijske funkcije treba biti izražena u radijanima. Kako mi ljudi koristimo zapis kuta u stupnjevima potrebno je napraviti pretvorbu u radijane prije primjene pojedine trigonometrijske funkcije.

Za računanje inverznih trigonometrijskih funkcija koristit ćemo izraze prikazane u tablici 5.6. U literaturi se često mogu naći različiti oblici zapisa ovih funkcija pa tako inverznu sinusnu funkciju pišemo kao $sin^{-1}(x)$ ili $arcsin(x)$ gdje je arc oznaka za arkus.

Funkcija	Kratica	Definicija
sinus	sin	nasuprotna kateta / hipotenuza
kosinus	cos	priležeća kateta / hipotenuza
tangens	tan (tg)	sin(x)/cos(x), nasuprotna kateta / priležeća kateta
kotangens	cot (ctg)	cos(x)/sin(x), priležeća kateta / nasuprotna kateta
sekans	sec	1/cos(x), hipotenuza / priležeća kateta
kosekans	csc	1/sin(x), hipotenuza / nasuprotna kateta

Tablica 5.4: Definicija trigonometrijskih funkcija.

Funkcija od x	Pseudokôd	Raptor	C♯
sinus	sin(x)	sin(x)	Math.Sin(x)
kosinus	cos(x)	cos(x)	Math.Cos(x)
tangens	tan(x)	tan(x)	Math.Tan(x)
kotangens	cot(x)	cot(x)	1/Math.Tan(x)
sekans	1/cos(x)	1/cos(x)	1/Math.Cos(x)
kosekans	1/sin(x)	1/sin(x)	1/Math.Sin(x)

Tablica 5.5: Zapisi trigonometrijskih funkcija u pseudokôdu, Raptoru i programskom jeziku C♯.

Funkcija od x	Pseudokôd	Raptor	C♯
arkus sinus	arcsin(x)	arcsin(x)	Math.Asin(x)
arkus kosinus	arcos(x)	arcos(x)	Math.Acos(x)
arkus tangens	arctan(x)	arctan(x)	Math.Atan(x)
arkus kotangens	arccot(x)	arccot(x)	Math.Atan(1/x)

Tablica 5.6: Zapisi inverznih trigonometrijskih funkcija u pseudokôdu, Raptoru i programskom jeziku C♯.

5.4 Operatori usporedbe

Operatori usporedbe ili relacijski operatori najčešće se koriste kao uvjeti prilikom grananja ili kao uvjeti pri kojima se završava izvođenje neke petlje. Operatorima usporedbe uspoređuju se vrijednosti pohranjene u dvije varijable, a za to je potrebno biti ispunjeno da su varijable istog tipa podataka. Kao izlaz, operator usporedbe generira logičku vrijednost, koja može biti jednaka nuli u slučaju ne ispunjenja relacije (npr. $1 < 0 = 0$) ili jedinici u slučaju da je operacija usporedbe točna (npr. $0 < 2 = 1$). Smatra se da vrijednost 0 označava neistinu odnosno laž, a vrijednost 1 istinu. Ako je potrebno napraviti usporedbu više od dvije varijable tada je potrebno koristiti logičke operatore da se poveže više usporedbi dviju

varijabli u jednu cjelinu. U tablici 5.7 prikazani su zapisi pojedinih relacijskih operatora u pseudokôdu, Raptoru i programskom jeziku C♯.

Opis	Pseudokôd	Raptor	C♯
Manje	<	<	<
Manje ili jednako	<=	<=	<=
Veće	>	>	>
Veće ili jednako	>=	>=	>=
Jednako	=	=	==
Različito	<>	<>	!=

Tablica 5.7: Zapisi operatora usporedbe u pseudokôdu, Raptoru i programskom jeziku C♯.

5.5 Logički operatori

Logički operatori također se mogu koristiti, u specijalnim slučajevima, kao dopuna operatorima usporedbe. Kao što je objašnjeno u poglavlju 4, logički operatori mogu se koristiti nad operandima koji su logičke vrijednosti, nula i jedan, ali mogu se koristiti i nad proizvoljnim brojevima, pod uvjetom da su oni zapisani u varijablama istog tipa. Logički operator I tako možemo koristiti kao logičko množilo, ILI kao nepotpuno zbrajalo, a isključivo ILI kao potpuno zbrajalo bez preljeva. Upravo pomoću logičkih operatora realiziraju se aritmetičke operacije u digitalnim računalima. Način zapisa pojedinih logičkih operatora u pseudokôdu, Raptoru i programskom jeziku C♯ prikazan je u tablici 5.8.

Opis	Pseudokôd	Raptor	C♯
NE	NE	NOT	!
I	I	AND	&&
ILI	ILI	OR	||
Isključivo ILI	XOR	XOR	^

Tablica 5.8: Zapisi logičkih operatora u pseudokôdu, Raptoru i programskom jeziku C♯.

5.6 Pravila za pisanje izraza

Pisanje matematičkih izraza u različitim programskih jezicima zahtjeva poznavanje sintakse i specifičnosti pojedinog programskog jezika. Pritom je važno znati

Opis	Sintaksa C♯
Uvećaj nakon, umanji nakon	$i++$, $i--$
Unarni operatori (+, -, !)	i += 5
Uvećaj prije, umanji prije	$++i$, $--i$
Množenje, dijeljenje (* / %)	a * b, a / b
Zbrajanje, oduzimanje	a + b, a - b
Operatori usporedbe (<, >, <=, >=)	a > b
Operatori jednakosti (==, !=)	a == b, a != b
Logički I	a && b
Logički ILI	a || b

Tablica 5.9: Prioritet izvođenja operacija u programskom jeziku C♯, od većeg prioriteta ka nižem, silazno.

koji je prioritet izvođenja neke operacije kako ne bismo dobili rezultat drugačiji od željenog zbog nepoštivanja prednosti izvođenja pojedine matematičke operacije. U matematici podrazumijevamo izvođenje operacije slijeva nadesno, osim u slučaju kada se dvije operacije različitog prioriteta nađu jedna pored druge. Tada se prvo izvodi operacija s višim prioritetom. Okruglim zagradama mogu se zaobići ugrađena pravila prioriteta izračunavanja izraza jer one imaju viši prioritet od svih operatora. U programskom jeziku C♯ prioritet matematičkih operacija prikazan je u tablici 5.9, na način da se prioritet izvođenja smanjuje odozgo prema dolje.

Zanimljivo je promotriti primjere zapisa složenijih matematičkih izraza, kod kojih je prioritet izvođenja operacije ključan za dobivanje ispravnog rješenja. Za primjere ćemo uzeti dvije jednadžbe koje sadrže tri varijable i napisane su u obliku razlomka ili složene potencije. Kako bi ih implementirali u programskom jeziku C♯ bit će potrebno poslužiti se zagradama, koje će nam omogućiti jednostavno dodjeljivanje prioriteta operacijama. U prvom primjeru prikazat ćemo ispravan i neispravan način zapis sljedećeg razlomka:

$$c = \frac{1}{a+b}. \tag{5.1}$$

Ispravan način zapisa razlomka 5.1 u programskom jeziku C♯, je:

```
// deklaracija varijabli
float a, b, c;

// izracun dijeljenja
c = 1 / (a + b);
```

Zagrade postavljaju prioritet izvođenja operacije zbrajanja ispred operacije dijeljenja, omogućavajući nam da prvo dobijemo zbroj vrijednosti u nazivniku, nakon čega se izvodi operacije dijeljenja jedinice sa zbrojenom vrijednošću. Neispravan način zapisa razlomka 5.1 u programskom jeziku C♯, je:

```
// deklaracija varijabli
float a, b, c;

// izracun dijeljenja
c = 1 / a + b;
```

Ovakvim neispravnim načinom zapisa dobiveni rezultat odgovarat će izrazu:

$$c = \frac{1}{a} + b. \tag{5.2}$$

Možemo općenito reći kako zagradama uvijek moramo odvajati operacije nižeg stupnja prioriteta, ako se one moraju izvesti prije operacija višeg stupnja prioriteta. Na primjeru potenciranja pokazat ćemo kako se nepažnjom pri uporabi zagrada dobiva rezultat koji ima sasvim drugu izračunatu vrijednost od željene. Definirajmo željenu funkciju kao:

$$f = a^{\frac{b+c}{d-e}}. \tag{5.3}$$

Ispravan način zapisa ove potencije s razlomkom u programskom jeziku C♯, je:

```
// deklaracija varijabli
float a, b, c, d, e;

// izracun potencije
f = Math.Pow(a, (b + c) / (d - e));
```

Neispravan način zapisa ove potencije s razlomkom u programskom jeziku C♯, je:

```
// deklaracija varijabli
float a, b, c, d, e;

// izracun potencije
f = Math.Pow(a, (b + c / d - e));
```

pa će ovakav zapis rezultirati izračunom funkcije oblika:

$$f = a^{b+\frac{c}{d}-e}. \tag{5.4}$$

5.7 Primjeri

U nastavku su dani primjeri pisanja naredbi u programskom jeziku C♯ za izračun pojedinih izraza. Primjeri koriste u ovom poglavlju objašnjene naredbe i metode za programiranje matematičkih izraza. Za ispis rezultat izračuna korištena je naredba "*Console.WriteLine()*" koja je detaljnije objašnjena u potpoglavlju 8.2. Programski kôd u pojedinom primjeru sastoji se od deklaracije varijabli, naredbi izraza i naredbe za ispis. Dio za unos moguće je dodati za vježbu nakon što se usvoji dio gradiva vezan za unos varijabli objašnjen u potpoglavlju 8.2.

■ Primjer 5.1

Za povećavanje vrijednosti neke varijable, najčešće brojača u petlji, koristimo varijablu "i". Što će se ispisati kao rezultat ako koristimo sintaksu $++i$, a što ako koristimo sintaksu $i++$?

Programski kôd 5.1: Razlikovanje inkrementiranja $++i$ od $i++$.

```
int i = 1;
Console.WriteLine(++i);
i = 1;
Console.WriteLine(i++);
```

Rješenje:

Definirana je varijabla "*i*" kojoj je pridružena vrijednost 1. Naredba "*Console.WriteLine(++i)*" ispisat će rješenje 2, što znači da je varijabli *i* prvo dodana jedinična vrijednost, a potom je očitana vrijednost varijable. Naredba "*Console.WriteLine(i++)*" ispisat će rješenje 1, što znači da je vrijednost varijable očitana, a tek naknadno se dodaje jedinična vrijednost varijabli.

■ Primjer 5.2

Pri definiranju varijabli možemo koristiti različite tipove podataka. Ovisno o izabranom tipu podatka, vrijednost pojedinih računskih operacija i sam prikaz rezultata računske operacije biti će različiti. Potrebno je odrediti što će se ispisati na zaslonu računala kao rezultat pojedine operacije u programskom jeziku C♯ nakon izvođenja programa 5.2.

Programski kôd 5.2: Primjer dijeljenja različitih tipova podataka.

```
// deklaracija i inicijalizacija varijabli
int a = 3;
int b = 2;
```

```
float c = 1.5;

// izracun i ispis varijabli
Console.WriteLine(a / b);
Console.WriteLine(a / c);
```

Rješenje:

Zadane su varijable "a", "b" i "c", od kojih su "a" i "b" cjelobrojne i zauzimaju 32 bita memorije, dok je varijabla "c" tipa float (s plivajućim zarezom) i zauzima također 32 bita memorije.

Naredba "*Console.WriteLine(a / b)*" dat će rezultat jednak 1 jer su obje varijable cjelobrojnog tipa pa je izvršena operacija cjelobrojnog dijeljenja s ostatkom, koji se ne prikazuje. Stvarni rezultat je 1 i ostatak 1. Naredba "*Console.WriteLine(a / c)*" dat će rezultat jednak 2 jer je varijabla "c" tipa float pa se izvršava operacija potpunog dijeljenja u preciznosti varijable float.

■ Primjer 5.3

Računalo kao ulaz dobiva duljinu dijagonale pravokutnika i kut između dijagonale i kraće stranice izražen u stupnjevima. Potrebno je, uporabom trigonometrijskih funkcija, napisati naredbe programa u programskom jeziku C♯ koje kao rezultat daju duljinu dulje stranice pravokutnika.

Rješenje:

Zadatak je moguće riješiti primjenom više trigonometrijskih funkcija, a u ovom slučaju poslužit ćemo se funkcijama sinus i tangens. U programskom jeziku C♯ implementirane su trigonometrijske funkcije u obliku pomoćnih sistemskih metoda "*Math.ImeFunkcije(x)*" gdje je ime "*ImeFunkcije*" najčešće istovjetno kratici funkcije koja se koristi pri pisanju matematičkih formula. U našem slučaju koristit će se pomoćne metode "*Math.Sin(x)*" i "*Math.Tan(x)*", gdje nam je "x" argument funkcije koji se unosi u radijanima.

Prema definiciji kosinus kuta je omjer priležeće katete i hipotenuze pravokutnog trokuta, u našem slučaju stranice i dijagonale pravokutnika. Pretpostavit ćemo da ulazne podatke imamo u varijablama pod nazivom "*kracaStr*", "*kutSt*" i "*dijagonala*". Rješenje možemo realizirati uporabom funkcija sinus i tangens pa ćemo izraz napisati za oba rješenja.

Programski kôd 5.3: Rješenje uporabom funkcija sinus i tangens.

```
// deklaracija varijabli
```

```
double rjesenje1, rjesenje2, dijagonala, kutSt, kracaStr;

// izracun
rjesenje1 = dijagonala * Math.Sin(kutSt * Math.PI / 180);
rjesenje2 = kracaStr / Math.Tan(kutSt * Math.PI / 180);
```

Važno je obratiti pažnju na to da kutove koji su zadani u stupnjevima moramo pretvoriti u radijane kako bi nam ugrađene funkcije dale dobar rezultat, a to činimo množenjem s konstantom π i dijeljenjem sa 180.

5.8 Zadaci za samostalan rad

Zadatak 5.1

Potrebno je postupkom cjelobrojnog dijeljenja prikazati matematički ispravan rezultat dijeljenja broja 17 *s brojem* 5. *Rezultat mora imati cjelobrojnu vrijednost i ostatak.*

Zadatak 5.2

Napisati funkciju kojom računamo logaritam po bazi 3 *iz broja* 27 *u Raptoru (dijagramu toka) i programskom jeziku C♯.*

Zadatak 5.3

Ako su zadane vrijednosti varijabli $a = 1$, $b = 0$, $c = 1$, *što će kao rezultat biti zapisano u varijabli* $d = (a + b) * c$*?*

Zadatak 5.4

*Matematički zapisati funkciju "c = Math.Sqrt(Math.Pow(a,2.0)+Math.Pow(b,2.0)-2*a*b*Math.Cos(g*Math.PI/180.0))".*

Poglavlje 6

Pseudokôd

Kako je već ranije navedeno, prilikom izrade programskog rješenja prvo se skicira ideja programskog rješenja. U tu svrhu najbolje je primijeniti tzv. pseudokôd, odnosno redoslijed naredbi programa napisati lako razumljivim riječima. Ideja je da tako napravljena skica programskog rješenja bude lako razumljiva i ljudima koji se do sada nisu susreli s nekim programskim jezikom. Tako se koriste oznake naredbi koje su lako razumljive onom tko je radio skicu mogućeg programskog rješenja, ali i svakom potencijalnom korisniku skice rješenja. Često su oznake u pripadnom materinjem jeziku programera ili se koristi engleski jezik. Napravljena skica programskog rješenja je iz tog razloga valjana i za svaki programski jezik. Prilikom izrade završnog programa naredbe pseudokôda zamijene se pripadnim naredbama ili grafičkim blokovima pojedinog programskog jezika. Kao primjer može se navesti pseudokôd 6.1 za jednostavni program koji uključuje samo ispis poruke, odnosno niza znakova, "*Hej svijete!*".

Pseudokôd 6.1: Ispis poruke "*Hej svijete!*".

Ulaz:
Izlaz:
Ispis: "Hej svijete!"

Primjer pseudokôda 6.1 ne prima nikakve ulazne vrijednosti za obradu (označeno s "**Ulaz:**") te ne vraća nikakvu izlaznu vrijednost (označeno kao "**Izlaz:**"). Jedina mu je funkcija ispis poruke na zaslon računala zadane kao niz znakova. Promatrajući pseudokôd 6.1 moguće je zaključiti kako neki programi, odnosno dijelovi programa koji se zovu i funkcije te pomoćne metode, mogu biti pozvani od nadređenog dijela programa da obrade proslijeđene podatke i vrate nadređe-

nom dijelu programa traženi rezultat. Tako u nekim programskim jezicima, kao što je to i C♯, razlikujemo glavnu metodu koju poziva operacijski sustav prilikom pokretanja programa te pomoćne metode koje onda poziva glavna metoda ili nadređene pomoćne metode. Odlika glavne metode je da vrlo često nema ulaznih i izlaznih varijabli dok to pomoćne metode vrlo često imaju.

6.1 Općenita struktura programa

Prilikom pisanja programa potrebno je poštivati strukturu koja odgovara načinu obrade podataka u računalu, a to znači slijedno izvršavanje naredbi. Sljedeća naredba u nizu izvodi se tek kada je prethodna izvedena pri čemu se koriste nove vrijednosti varijabli dobivene izvršavanjem prethodne naredbe. Za spremanje podataka tijekom izvođenja programa koriste se varijable, koje se moraju prvo deklarirati odnosno stvoriti. Također se preporuča inicijalizacija varijabli (pridruživanje početne vrijednosti) prije njihovog prvog korištenja.

U svakom programskom jeziku, odnosno u njegovoj razvojnoj okolini postoji tzv. preambula u sklopu koje se rade podešavanja i uključivanja različitih biblioteka s gotovim funkcijama ili pomoćnim metodama. Preambula ovisi o programskom jeziku i ovdje će se pretpostaviti kako je razvojna okolina već stvorila potrebne postavke u sklopu odabranog predloška programa. Biblioteke s gotovim funkcijama ili pomoćnim metodama izrađuje proizvođač razvojne okoline za izradu aplikacija ili programer sam izrađuje svoju biblioteku. Sam programski kôd se piše u sklopu glavne metode i općenita struktura programa se sastoji od sljedećih dijelova:

Deklaracija varijabli - je uvijek prvi dio programa koji služi za pobrojavanje svih potrebnih varijabli u programu. Kod deklaracije se navodi tip i ime varijable korištenjem ključnih riječi pojedinog programskog jezika;

Inicijalizacija varijabli - je drugi preporučljivi dio programa, a služi kako bi se deklariranim varijablama pridružile početne vrijednosti. Inicijalizacija varijabli je potrebna jer se prilikom deklaracije varijabli zauzima potrebna memorija, a u zauzetoj memoriji mogu biti vrijednosti zaostale od prošlih obrada podataka odnosno varijabli. Današnji viši programski jezici brišu sadržaj memorije prilikom deklaracije varijable, no to nije uvijek slučaj;

Unos podataka - je treći dio programa koji služi za dohvat (unos) podataka koji će se obraditi u programu. Ovdje je potrebno imati na umu da računalo obrađuje podatke koji su u memoriji pa ih je prije obrade potrebno i unijeti u memoriju te spremiti u pripadne varijable. Upravo to se događa u ovom dijelu programa. Podaci koje unosi operater se pridružuju varijablama

koje se kasnije koriste/obrađuju u programu. Korištenje varijable kojoj nije prethodno pridružena vrijednost uzrokuje pogrešku prilikom izvođenja programa, na što je potrebno uvijek obratiti pažnju;

Obrada podataka - predstavlja četvrti dio programa u kojemu se na osnovi unesenih podataka obradom dobivaju novi podaci, odnosno program izračunava traženi rezultat. Ti novi podaci se uvijek spremaju u pripadne varijable kako bi se mogli koristiti za kasniju obradu. Treći i četvrti dio programa (unos i obrada podataka) često su međusobno povezani i kod velike količine podataka se ponavljaju u izvođenju. Kao primjer možemo navesti gledanje filma s DVD-a, gdje se tijekom cijelog vremena gledanja filma ponavlja korak unosa podataka (čitanje podataka s DVD-a) te korak obrade podataka (dekôdiranje videozapisa i njegovo prosljeđivanje izlaznoj jedinici odnosno zaslonu);

Ispis rezultata - predstavlja peti i zadnji dio općenite strukture programa. U njemu se radi ispis rezultata dobivenih tijekom obrade podataka. Za ispis vrijednosti spremljene u pojedinoj varijabli se koristi jedna od izlaznih jedinica računala. Vrlo često se rezultati obrade ispisuju na zaslon računala.

Pojedine dijelove općenite strukture programa moguće je promotriti u primjeru zbrajanja dva broja danog pseudokôdom 6.2. Kao pomoć u razumijevanju danog pseudokôda moguće je iskoristiti svima poznati način rada digitalnog kalkulatora. Na njemu se računanje zbroja dva broja provodi na način da se prvo unese prvi pribrojnik, zatim se odabere operacija zbrajanja, unese se drugi pribrojnik i pokrene se postupak izračuna pritiskom na tipku "=". Navedena procedura u biti predstavlja slijed naredbi da se digitalni kalkulator isprogramira da izračuna sumu dva broja, odnosno način programiranja izračune zbroja dva broja. Analogno tome pseudokôd 6.2 predstavlja proceduru odnosno slijed naredbi u računalu. Opet je dana glavna metoda bez potrebe za ulaznim i izlaznim varijablama. U programu se prvo radi deklaracija tri potrebne varijable u koje će se spremiti rezultat zbroja i dva pribrojnika. Ovdje je potrebno primijetiti da su imena varijabli odabrana tako da je značenje varijabli jasno. Zatim se sve tri varijable inicijaliziraju na početnu vrijednost 0. U jednoj naredbi je moguće inicijalizirati na istu početnu vrijednost i više varijabli. Potrebno je samo varijable nanizati korištenjem operatora pridruživanja i na kraju dati željenu početnu vrijednost. Sada se radi unos potrebnih podataka odnosno dva pribrojnika koje želimo zbrojiti. Također je jednom naredbom moguće unijeti vrijednosti za više varijabli pri čemu se varijable međusobno odvajaju zarezom. Prva vrijednost se sprema u prvu varijablu, druga vrijednost u drugu varijablu i tako redom. Nakon unosa dva pribrojnika u memoriji računala nalaze se svi potrebni podaci za obradu i moguće je odrediti iznos zbroja naredbom koja va-

Pseudokôd 6.2: Zbrajanje dva broja.

Ulaz:
Izlaz:
Deklaracija:
zbroj, prviPribrojnik, drugiPribrojnik
Inicijalizacija:
zbroj := prviPribrojnik := drugiPribrojnik := 0
Unos:
prviPribrojnik, drugiPribrojnik
Izračunaj:
zbroj := prviPribrojnik + drugiPribrojnik
Ispis:
zbroj

rijabli "*zbroj*" pridružuje rezultat operacije zbrajanja unesenih pribrojnika. Na kraju se radi ispis rezultata, odnosno vrijednosti u varijabli "*zbroj*".

6.2 Pravila pisanja pseudokôda

Prilikom pisanja pseudokôda postoje pravila kojih se je potrebno pridržavati kao i u slučaju viših programskih jezika. Za pojedine karakteristične načine obrade podataka kao što su grananja i petlje postoje definirane ključne riječi koje omogućavaju lakše razumijevanje programa. Usporedni pregled osnovnih ključnih riječi u pseudokôdu i pripadnih ključnih riječi u programskom jeziku C♯ dan je u tablici 6.1.

U tablici 6.1 moguće je vidjeti kako naredbe grananja i petlji sadrže vlastiti programski odsječak, odnosno programski blok. Kako bi se vizualno moglo lakše razlučiti koje naredbe spadaju u pojedini programski blok, koriste se uvlake. To je najbolje vidljivo kod grananja i petlji. Uvlake se koriste u sprezi s oznakom početka i završetka programskog bloka. Kako je pseudokôd slobodniji stil za skiciranje ideje programskog rješenja, često se zbog jednostavnosti ispuštaju oznake početka i završetka programskog bloka te se koriste samo uvlake. Ako se kod grananja i svih petlji prikazanih u tablici 6.1 koristi više naredbi u pripadnom programskom bloku tada se u programskom jeziku C♯ trebaju uvijek koristiti oznake početka i završetka programskog bloka. To se radi vitičastim zagradama pa je tako početak programskog bloka označen s "{", a završetak s "}". Po definiciji je prva naredba nakon grananja ili početka petlje automatski pridružena programskom bloku grananja ili petlje u slučaju da nema oznake početka i završetka programskog bloka. Ako su pak u programskom bloku po-

Opis	Pseudokôd	C♯

Programski blok	{
	naredbe
	}
		{
		naredbe;
		}

Unos podataka	**Unos:**	Console.Read(); Console.ReadKey(); Console.ReadLine();
Ispis podataka	**Ispis:**	Console.Write(); Console.WriteLine();
Pridruživanje vrijednosti	:=	=

Grananje	**ako je** uvjet **onda**
	naredba1
	inače ako je uvjet **onda**
	naredba2
	inače
	naredba3
		if (uvjet)
		naredba1;
		else if (uvjet)
		naredba2;
		else
		naredba3;

Petlja while	**dok je** uvjet **činiti** naredba	while (uvjet) naredba;
Petlja do while	**ponavljati** naredba **do** uvjet	do naredba; while (uvjet);
Petlja for	**za** i:=p **do** k **činiti** naredba	for (i=p;i<=k;i++) naredba;

Tablica 6.1: Pregled ključnih riječi za osnovne operacije u pseudokôdu te programskom jeziku C♯.

trebne dvije ili više naredbi, uvode se oznake početka i završetka programskog bloka. Više detalja o korištenju uvlaka te oznaka početka i završetka programskog bloka može se pronaći u poglavlju 8 koje se odnosi na programski jezik C♯.

6.3 Unos i ispis podataka

Naredbe za unos i ispis podataka omogućuju razmjenu podataka između programa i njegove okoline. Okolinu programa predstavljaju drugi programi i operater koji koristi program odnosno samo računalo. Moguća razmjena podataka je dvosmjerna, a referenca za određivanje smjera je sam program. To znači da se pod unosom pretpostavlja smjer podataka od operatera prema programu. Radi se o podacima koji su potrebni programu za obradu. Pod ispisom se smatra

smjer podataka od programa prema operateru. Radi se o podacima dobivenim u programu tijekom obrade podataka, odnosno rezultati izvršavanja programa potrebni operateru.

Kao što je vidljivo u pseudokôdu 6.2, za unos podataka je potrebno samo navesti imena varijabli u koje će se podaci spremiti. Unos podataka moguće je raditi na više mjesta tijekom izvršavanja programa kako je navedeno u objašnjenju općenite strukture programa. Kod unosa podataka bitno je poštivati različite tipove podataka što znači da se podatak tipa cijeli broj može spremiti samo u numeričke varijable koje omogućavaju obradu (spremanje) cjelobrojnih numeričkih vrijednosti. To je naročito bitno za pisanje programa kod nekih viših programskih jezika. U slučaju pseudokôda se tip podatka može ispustiti uz pretpostavku da će ga pripadna varijabla prilikom pridruživanja automatski prepoznati i ispravno spremiti u memoriju. Na tom principu rade neki alati za matematičko modeliranje i simuliranje kao što je MATLAB pa se takva pretpostavka može napraviti.

Kod ispisa podataka mogu se ispisivati poruke radi olakšanja rada s programom te vrijednosti pojedinih varijabli. Za ispisivanje poruka koristi se tip podataka za prikaz niza znakova, odnosno "*string*". Kako bi se mogle ispisivati vrijednosti varijabli unutar poruke koristi se operator nadovezivanja nizova znakova simbola "+". U slučaju nizova znakova operacija zbrajanja nizova znakova provodi se tako da se nizovi znakova dodaju jedan iza drugoga, odnosno provodi se proces konkatenacije. Rezultat operatora spajanja nizova znakova se najbolje vidi u analizi primjera za izračun najveće brzine cestovne dionice koja se još uvijek nalazi u području tolerancije. Skica pripadnog programskog rješenja dana je pseudokôdom 6.3. Ideja je odrediti najveću dopuštenu brzinu dionice cestovne prometnice na kojoj postoji ograničenje brzine, a da je ona još uvijek u području tolerancije. To područje tolerancije često se izražava u postocima i obično iznosi oko 10%. Kako bi se tražena vrijednost izračunala, potrebni su podaci vezani za iznos ograničenja te postotni iznos tolerancije. Za ideju programskog rješenja potrebno je deklarirati varijable za te dvije vrijednosti i naravno rezultat kako je prikazano u pseudokôdu 6.3. Nakon deklaracije varijabli radi se unos podataka. U ovom slučaju je unos podataka povezan s ispisom odgovarajućih poruka koji operateru olakšavaju rad s programom. Poruka je zadana kao niz znakova i objašnjava koji podatak operater treba unijeti u program. Potrebno je primijetiti kako se prvo ispiše poruka, a zatim se čeka na unos podatka. Operater obavještava program da je završio unos traženog podatka pritiskom na tipku "*Enter*". Kada je završen unos svih potrebnih podataka, izračunava se najveća brzina uz pretvaranja unesenih postotaka u bezdimenzionalnu vrijednost. Zadnja naredba pseudokôda je ispis dobivenog rezultata. Konačni niz znakova koji će se ovdje ispisati sastoji se od dva dijela. Prvi dio je fiksna poruka zadana kao niz znakova, a drugi dio je promjenjiva vrijednost zadana u obliku varijable.

Pseudokôd 6.3: Izračun najveće dopuštene vrijednosti brzine ograničenja.

Ulaz:
Izlaz:
Deklaracija:
iznosOgranicenja, postotakTolerancije, najvecaBrzina
Inicijalizacija:
iznosOgranicenja:=postotakTolerancije:=najvecaBrzina:=0
Ispis:
"Unesite iznos ograničenja brzine >"
Unos:
iznosOgranicenja
Ispis:
"Unesite postotak tolerancije ograničenja brzine >"
Unos:
postotakTolerancije
Izračunaj:
najvecaBrzina:=iznosOgranicenja*(1+postotakTolerancije/100)
Ispis:
"Najveća dopuštena brzina iznosi " + najvecaBrzina

Dijelovi se međusobno spajaju u jedan niz znakova korištenjem prije objašnjenog operatora "+". Prilikom ispisa se numerička vrijednost spremljena u varijabli "*najvecaBrzina*" automatski pretvara u pripadni niz znakova za ispis. Analogno radi i ispis u programskom jeziku C♯. U slučaju da se poruka za ispis sastoji od više dijelova, oni se samo spajaju operatorom "+". Potrebno je primijetiti da se poruka uvijek ispisuje doslovno, odnosno sve potrebne razmake (bjeline) moramo dodati u fiksni dio poruke.

6.4 Grananja

Kako je ranije navedeno, programi odnosno naredbe se načelno izvršavaju slijedno. To znači jedna za drugom kako su navedene u programskom kôdu. Tijekom izvršavanja programa ponekad se javlja potreba za provjerom određenog uvjeta i prema vrijednosti rezultata provjere (postavljeni uvjet je ispunjen ili nije ispunjen) se izvrši određeni programski kôd. U tu se svrhu koristi struktura grananja koja omogućuje izvršavanje određenog programskog kôda ovisno o ispunjenosti zadanog uvjeta. Pri tome se programski kôd može sastojati samo od jedne naredbe ili od više naredbi. U slučaju da se sastoji od više naredbi nazivamo ga i programskim blokom. Postoji nekoliko vrsta grananja od kojih

najjednostavnija omogućava provjeru uvjeta za izvršavanje programskog kôda, a složenija struktura grananja omogućuje uzastopnu provjeru više uvjeta. U nastavku će se opisati sve strukture grananja počevši od najjednostavnije.

6.4.1 Definiranje uvjeta za grananje

Kako bi se moglo izvesti grananje unutar programa, potrebno je definirati uvjet koji će struktura grananja ispitati. Rezultat ispitivanja uvjeta je uvijek logička vrijednost 0 (uvjet nije ispunjen) ili 1 (uvjet ispunjen), a slaže se pomoću kombinacije logičkih, relacijski i aritmetičkih operatora opisanih u prethodnom poglavlju. Smatra se da je uvjet ispunjen (istinit) ako je njegov rezultat jednak logičkoj 1, odnosno da nije ispunjen (neistinit) ako je njegov rezultat jednak logičkoj 0. Kao uvjet se može zadati i numerička vrijednost pri čemu vrijedi pravilo da numerička vrijednost nula odgovara logičkom 0, a svaka druga numerička vrijednost odgovara logičkom 1.

Općenito se uvjet grananja sastoji od aritmetičkog (numeričkog) i logičkog dijela. Takav uvjet predstavlja složeni logički izraz za čije je izvršavanje, odnosno definiranje, potrebno u obzir uzeti sljedeća pravila:

- Aritmetički dio uvjeta ima veći prioritet (izračunava se prvi);
- Primjenom zagrada moguće je prilagoditi prioritet izvršavanja;
- Za spajanje više relacijskih operatora u jedan uvjet koriste se logički operatori;
- Relacijski operatori se primjenjuju na jedan par varijabli ili izraza;
- Logički operatori imaju najniži prioritet (izračunavaju se zadnji);
- Kod logičkih operatora najveći prioritet ima unarni operator NE;
- Rezultat uvjeta se izračunava u dijelovima ovisno o prioritetu.

6.4.2 Grananje "*ako je*" ("*if*")

Ovo grananje predstavlja najjednostavnije grananje. Moguće ga je objasniti pomoću pseudokôda 6.4. Dani pseudokôd je podijeljen na tri dijela. U prvom dijelu se izvode naredbe programa dok na red ne dođe prva naredba drugog dijela odnosno ispitivanje uvjeta. Uvjet se zadaje kao logička relacija ili kao usporedba numeričkih vrijednosti. U slučaju da je uvjet istinit odnosno ispunjen izvršavaju se naredbe grananja. Nakon njih se nastavlja s izvršavanjem programa, odnosno izvodi se prva naredba nakon strukture grananja. Ako uvjet nije ispunjen, naredbe grananja se preskaču i kao sljedeća naredba izvršava se prva naredba koja dolazi nakon strukture grananja, odnosno treći dio pseudokôda. Radi lakšeg vizualnog praćenja pripadnosti naredbi se naredbe koje pripadaju strukturi

grananja pišu uz uvlaku prema ostalim naredbama. Za oznaku kraja strukture grananja se koristi ključna riječ "*kraj ako*".

Pseudokôd 6.4: Grananje "*ako je*" ("*if*").

naredbe prije grananja
ako je uvjet **onda**
  naredbe grananja
kraj ako
naredbe nakon grananja

Primjena grananja "*ako je*" ("*if*") može se objasniti na primjeru izračuna apsolutne vrijednosti unesenog broja. Prema matematičkoj definiciji apsolutna vrijednost broja uvijek je pozitivna i jednaka je unesenom broju ako je on pozitivan, odnosno u slučaju negativnog broja jednaka je tom broju s pozitivnim predznakom. Jednostavan program za izračun apsolutne vrijednosti dan je pseudokôdom 6.5. Na početku danog pseudokôda deklarira se varijabla "*podatak*" potrebna za unos numeričke vrijednosti te izračun apsolutne vrijednosti. Nakon unosa provjerava se je li vrijednost varijable "*podatak*" manja od nule. U slučaju da jest, izvršava se naredba grananja koja mijenja predznak varijable "*podatak*" pretvarajući ju u pozitivnu prema matematičkoj definiciji apsolutne vrijednosti. Nakon toga se izvršava naredba za ispis rezultata. U slučaju da uvjet nije ispunjen, odnosno varijabla "*podatak*" je veća ili jednaka nuli, odmah se izvršava naredba za ispis rezultata. Moguće je primijetiti kako se u ovom slučaju preskače naredba vezana za strukturu grananja, odnosno uvučena naredba za promjenu predznaka.

Pseudokôd 6.5: Izračun apsolutne vrijednosti.

Ulaz:
Izlaz:
Deklaracija:
    podatak
Ispis:
    "Unesite broj za izračun apsolutne vrijednosti >"
Unos:
    podatak
ako je podatak < 0 **onda**
  podatak := - podatak
kraj ako
Ispis:
    "Apsolutna vrijednost unesenog broja iznosi "+ podatak

6.4.3 Grananje "*ako je - inače*" ("*if - else*")

U nekim slučajevima potrebno je izvršiti određeni programski blok i kada uvjet nije ispunjen, prije nego se nastavi s izvršavanjem programa. Tada se koristi struktura grananja "*ako je - inače*" ("*if - else*") koja sadrži opciju zadavanja programskog bloka za slučaj da uvjet nije ispunjen. Ovo grananje je definirano pseudokôdom 6.6. Dani pseudokôd može se podijeliti u četiri dijela. U prvom se dijelu izvode naredbe programa dok na red ne dođe prva naredba drugog dijela programa, odnosno ispitivanje uvjeta grananja. U slučaju da je uvjet istinit (ispunjen) izvršavaju se naredbe za slučaj da je uvjet ispunjen. Nakon njih se dalje izvršava prva naredba iz zadnjeg četvrtog dijela koji dolazi nakon strukture grananja. Ovaj slučaj analogan je onom kod prethodne inačice grananja. Pri tome se blok naredbi vezan za slučaj da uvjet nije ispunjen preskaču i ne izvode se. U slučaju da uvjet nije ispunjen, izvode se naredbe programskog bloka vezane uz ključnu riječ "*inače*", odnosno naredbe za slučaj da uvjet nije ispunjen. Nakon toga se izvršava prva naredba iz zadnjeg četvrtog dijela koji dolazi nakon grananja. Pri tome se sada naredbe programskog bloka vezanog za slučaj da je uvjet ispunjen preskaču i ne izvode se.

Pseudokôd 6.6: Grananje "*ako je - inače*" ("*if - else*").

```
naredbe prije grananja
ako je uvjet onda
   naredbe uvjet ispunjen
inače
   naredbe uvjet nije ispunjen
kraj ako
naredbe nakon grananja
```

Primjena "*ako je - inače*" ("*if - else*") grananja može se objasniti na primjeru provjere premašuje li neka veličina iznos zadanog praga. Program za rješenje ovog problema prikazan je pseudokôdom 6.7. U prvom dijelu rješenja deklariraju se potrebne varijable (jedna varijabla za spremanje praga te druga za spremanje vrijednosti koja se ispituje) i unose potrebne vrijednosti varijabli. Nakon unosa vrijednosti obje varijable ispituje se je li vrijednost varijable "*podatak*" manja ili jednaka od unesenog praga. Ako je taj uvjet ispunjen ispisuje se poruka da unesena vrijednost ne prekoračuje prag i nakon toga program završava. U slučaju da uvjet nije ispunjen ispisuje se poruka da unesena vrijednost prekoračuje prag i nakon toga program završava. Potrebno je primijetiti da se u svakom slučaju ispisuje samo jedna poruka dok se druga poruka preskače, odnosno izvodi se samo jedan programski blok iz strukture grananja od moguća dva.

Pseudokôd 6.7: Provjera prekoračenja praga.

```
Ulaz:
Izlaz:
Deklaracija:
        podatak, prag
Ispis:
        "Unesite vrijednost praga >"
Unos:
        prag
Ispis:
        "Unesite vrijednost za provjeru prekoračenja praga >"
Unos:
        podatak
ako je podatak <= prag onda
  Ispis:
          "Unesena vrijednost ne prekoračuje prag."
inače
  Ispis:
          "Unesena vrijednost prekoračuje prag."
kraj ako
```

6.4.4 Grananje "*ako je - inače ako je - inače*" ("*if - else if - else*")

U nekim slučajevima potrebno je ispitati više uvjeta vezanih za istu skupinu varijabli pa je potrebno imati na raspolaganju više od dva programska bloka. Primjer je odabir najboljeg (najmanjeg) cestovnog vozila za prijevoz tereta. Općenito je u takvom slučaju na raspolaganju više vrsta cestovnih vozila, od motocikla do teretnog vozila s prikolicom za glomazne terete. Svako od vozila ima definiranu najveću nosivost te najveće dopuštene dimenzije tereta. Za odabir je potrebno redom ispitivati u koje područje dani teret spada jer je za optimalan transport odnosno najjeftiniji prijevoz potrebno uzeti najprikladnije vozilo.

Takvi problemi rješavaju se ispitivanjem više uvjeta uzastopce. Pri tome broj dodatnih uvjeta ovisi o pripadnom problemu. Struktura grananja koja omogućuje ispitivanje više uvjeta uzastopce dana je pseudokôdom 6.8. Potrebno je naglasiti kako su uvjeti u ovoj strukturi grananja vezani za postizanje istog rezultata. Svaki uvjet koji se ispituje ima pridruženi pripadni programski blok koji se izvodi samo ako je pripadni uvjet ispunjen. Također je potrebno izraditi i programski blok koji će se izvršiti ukoliko niti jedan od uvjeta nije ispunjen. U pseudokôdu 6.8 su kao primjer dana tri uvjeta, a može ih biti po volji mnogo.

Prvi dio danog pseudokôda odnosi se na uvijek prisutne naredbe deklaracije i

inicijalizacije varijable te unosa podataka. Zatim se dolazi do naredbe grananja i ispituje se prvi uvjet. Ako je on ispunjen, izvodi se pripadni programski blok i zatim se izvršava prva naredba koja dolazi nakon cijele strukture grananja. U slučaju da prvi uvjet nije ispunjen, ispituje se drugi uvjet. Ako je drugi uvjet ispunjen, izvršava se programski blok pridružen drugom uvjetu. Nakon toga se izvršava prva naredba koja dolazi nakon cijele strukture grananja. Ukoliko drugi uvjet nije ispunjen pristupa se provjeri sljedećeg uvjeta, odnosno u ovom slučaju trećeg uvjeta. Ako je on ispunjen, izvršava se programski blok pridružen njemu. Nakon njega se izvršava prva naredba koja dolazi nakon cijele strukture grananja. U slučaju da ni treći uvjet nije ispunjen izvodi se programski blok pridružen ključnoj riječi "*inače*". Nakon što se on izvršio, program nastavlja izvršavanjem prve naredbe koja dolazi nakon cijele strukture grananja.

Pseudokôd 6.8: Grananje "*ako je - inače ako je - inače*" ("*if - else if - else*").

naredbe prije grananja
ako je prvi uvjet **onda**
  naredbe prvi uvjet ispunjen
inače ako je drugi uvjet **onda**
  naredbe drugi uvjet ispunjen
inače ako je treći uvjet **onda**
  naredbe treći uvjet ispunjen
inače
  naredbe niti jedan uvjet nije ispunjen
kraj ako
naredbe nakon grananja

Primjena "*ako je - inače ako je - inače*" ("*if - else if - else*") grananja se može objasniti na primjeru provjere je li neki broj veći, manji ili jednak nuli. Program za rješenje ovog problema mora napraviti dvije provjere. Prvo provjeru je li broj manji od nule za utvrđivanje negativnog broja, zatim provjeru je li veći od nule za utvrđivanje pozitivnog broja, a ako niti jedan od ova dva uvjeta nije ispunjen moguće je zaključiti kako je broj jednak nuli. Programsko rješenje ovog problema dano je pseudokôdom 6.9. Na početku se klasično deklarira varijabla "*podatak*" i unosi joj se vrijednost uz ispis pripadne poruke. Zatim se ispituje prvi uvjet kako bi se ustanovilo je li vrijednost varijable "*podatak*" manja od nule. Ako je ovaj prvi uvjet ispunjen, ispisuje se poruka koja operateru javlja kako je unesena vrijednost manja od nule. Nakon toga program završava s radom. U slučaju da ovaj uvjet nije ispunjen program nastavlja s ispitivanjem drugog uvjeta kako bi se ustanovilo je li vrijednost varijable "*podatak*" veća od nule. Ako je sada ovaj drugi uvjet ispunjen, ispisuje se poruka koja operateru javlja da je unesena vrijednost veća od nule. Nakon toga program završava s radom. U slučaju da

ni ovaj drugi uvjet nije ispunjen, izvršava se naredba vezana uz ključnu riječ "*inače*", jer su svi uvjeti provjereni i niti jedan nije bio ispunjen. Ovaj slučaj vezan je za vrijednost varijable "*podatak*" jednake nuli. Ispisuje se poruka koja označava da je vrijednost varijable "*podatak*" jednaka nuli. Nakon izvršavanja ovog ispisa, program završava jer nema naredbi nakon strukture grananja.

Pseudokôd 6.9: Ispitivanje predznaka broja.

Ulaz:
Izlaz:
Deklaracija:
    podatak
Ispis:
    "Unesite vrijednost za provjeru predznaka >"
Unos:
    podatak
ako je podatak < 0 **onda**
  Ispis:
    "Unesena vrijednost je negativna."
inače ako je podatak > 0 **onda**
  Ispis:
    "Unesena vrijednost je pozitivna."
inače
  Ispis:
    "Unesena vrijednost je jednaka nuli."
kraj ako

6.4.5 Skretnica (grananje "*switch*")

Opisane strukture grananja omogućuju ispitivanje uvjeta pri čemu uvjet ne mora biti nužno vezan za vrijednost iste varijable. Ponekad je potrebno pojedini skup naredbi izvršiti za točno određenu vrijednost varijable čija se vrijednost provjerava. U takvim slučajevima primjenjuje se skretnica (grananje "*switch*") koja je jednostavnija za implementaciju rješenja takvih problema. Svrha skretnice je dopustiti vrijednosti varijable ili rezultata nekog izraza upravljanje tokom izvršavanja programa. Koncept skretnice u programiranju je dan pseudokôdom 6.10. Vidljivo je kako je skretnica vezana uz vrijednost varijable deklarirane i obrađene prije pozivanja skretnice. Sama skretnica radi na način da ispituje vrijednost pripadne varijable u pojedinom slučaju. Svaki pojedini slučaj se odnosi na određenu vrijednost ispitivane varijable. Pojedini slučaj se zadaje u obliku konstante prilikom izrade programa, odnosno predstavlja jednu stalnu vrije-

Pseudokôd 6.10: Skretnica (grananje "*switch*").

```
naredbe prije skretnice
skretnica (varijabla)
        slučaj prva vrijednost:
                naredbe za prvu vrijednost
        slučaj druga vrijednost:
                naredbe za drugu vrijednost
        zadano:
                naredbe za nedefiniranu vrijednost
kraj skretnica
naredbe nakon skretnice
```

dnost koja se ne mijenja tijekom izvršavanja programa. Broj slučajeva ovisi o pojedinom problemu koji je potrebno riješiti i može ih biti po volji mnogo. Za sigurno izvršavanje programa za vrijednost ispitivane varijable, koja nije definirana u pojedinom slučaju, koriste se naredbe pridružene slučaju označenim ključnom riječi "*zadano*". Kraj strukture skretnice se označava ključnom riječi "*kraj skretnica*".

Primjena skretnice može se objasniti na primjeru ispisa simboličke (tekstualne) oznake pojedine ocjene. U Republici Hrvatskoj ocjene se u školskom sustavu zadaju kao cjelobrojne vrijednosti u rasponu od 1 do 5, odnosno simbolički od nedovoljan do izvrstan. Program koji rješava ovaj problem dan je pseudokôdom 6.11. Na početku pseudokôda deklarira se varijabla "*ocjena*" i unosi joj se vrijednost. Nakon toga se poziva skretnica vezana uz varijablu "*ocjena*". Kako ocjena može poprimiti samo jednu od pet vrijednosti iz skupa $\{1,2,3,4,5\}$ definirano je pet slučajeva. Svaki slučaj se odnosi na jednu od mogućih numeričkih vrijednosti ocjene. Naredba u svakom slučaju je ispis simboličke oznake pojedine ocjene. Prilikom unosa ponekad dođe do pogreške što znači da je potrebno osigurati sigurno izvršavanje programa i u ovom slučaju. Iz tog razloga je slučaju skretnice označenom ključnom riječi "*zadano*" pridružena naredba koja ispisuje poruku operateru, odnosno korisniku programa da nije unio valjanu numeričku vrijednost koja označava ocjenu.

Prilikom izvršavanja skretnice iz memorije računala uzima se vrijednost pridružena varijabli "*ocjena*" i traži se slučaj koji odgovara njenoj vrijednosti. Ukoliko skretnica sadrži takav slučaj, naredbe pridružene tom slučaju se izvrše i nakon toga se izvršava prva naredba nakon cijele strukture skretnice. Ako nije pronađen slučaj skretnice koji odgovara vrijednosti varijable "*ocjena*" izvršava se naredba uz ključnu riječ "*zadano*". Nakon toga program nastavlja izvršavanjem prve naredbe nakon cijele strukture skretnice. Kako nakon skretnice ovaj program nema više daljnjih naredbi on završava.

Pseudokôd 6.11: Ispis simboličke ocjene.

```
Ulaz:
Izlaz:
Deklaracija:
        ocjena
Ispis:
        "Unesite numeričku vrijednost ocjene >"
Unos:
        ocjena
skretnica (ocjena)
        slučaj 1:
                Ispis:
                        "Nedovoljan"
        slučaj 2:
                Ispis:
                        "Dovoljan"
        slučaj 3:
                Ispis:
                        "Dobar"
        slučaj 4:
                Ispis:
                        "Vrlo dobar"
        slučaj 5:
                Ispis:
                        "Izvrstan"
        zadano:
                Ispis:
                        "Unesena vrijednost nije ocjena"
kraj skretnica
```

6.5 Petlje

Računala se najčešće primjenjuju za obradu velike količine podataka. Pri tome se postupak obrade pojednostavljuje i vrlo često se svaki podatak iz skupine obrađuje po istom načelu odnosno istim naredbama. Kao primjer može se uzeti naplata cestarine. Prilikom naplate operater u naplatnoj kućici za svako vozilo ponovi sljedeće radnje: (i) pozdravi vozača; (ii) identificira vrstu vozila i preuzme karticu cestarine; (iii) priopći vozaču iznos cestarine; (iv) preuzme novac ili bankovnu karticu od vozača; (v) naplati cestarinu; (vi) vrati vozaču račun i ostatak novca ili bankovnu karticu; te (vii) zaželi vozaču sretan put. Tih sedam

radnji može se shvatiti kao sedam naredbi za računalo koje bi obavljalo naplatu cestarine. Za naplatu cestarine u takvom se računalu može iskoristiti tzv. petlja koja će svih sedam naredbi ponoviti za svako vozilo koje dođe. Naredbe čine jedan programski blok koji se u ovom slučaju naziva tijelo petlje. Naplata cestarine za jedno vozilo predstavlja jedno izvršavanje tijela petlje što se naziva jedna iteracija. Za izvršavanje tijela petlje potrebno je da bude ispunjen uvjet petlje analogno kao kod grananja. Razlika je što se u ovom slučaju tijelo petlje izvršava ponovno tako dugo dok je uvjet ispunjen (istinit). Na taj način je moguće cestarinu naplatiti korištenjem računala, odnosno automatski bez operatera naplate. Tek nakon što uvjet za izvršavanje petlje više nije istinit petlja završava i izvodi se prva naredba programa nakon cijele strukture petlje.

Postoje tri vrste petlje ovisno o načinu pripreme podataka za petlju, mjestu provjere uvjeta te načinu promjene kontrolne varijable koja se provjerava u uvjetu petlje. Razlikuje se tako petlja s provjerom uvjeta na početku (petlja "*dok je*" odnosno "*while*"), petlja s provjerom uvjeta nakon izvršavanja tijela petlje (petlja "*ponavljati - do*" ili "*do - while*") te petlja s poznatim brojem ponavljanja (petlja "*za - do*" ili "*for*"). Viši programski jezici imaju definirane sve tri vrste petlji. Razlikuje se samo pravila zapisa petlje (sintaksa) te ključne riječi. Svaki problem, koji zahtjeva primjenu petlje za njegovo rješavanje, može se riješiti bilo kojom od navedene tri vrste petlji. Odabir često ovisi o preferenciji programera ili zahtjevu naručitelja. U nastavku je svaka od navedene tri petlje detaljnije objašnjena.

6.5.1 Petlja "*dok je*" ("*while*")

Petlja "*dok je*" ("*while*") ispituje uvjet prije izvršavanja tijela petlje. Tijelo petlje će se izvesti samo ako je uvjet ispunjen. Kod ispitivanja uvjeta koriste se varijable obrađene u programskom kôdu prije početka strukture petlje. To znači kako postoji mogućnost da se tijelo petlje neće izvršiti niti jedanput. Formalno zapisano tijek petlje "*dok je*" ("*while*") je sljedeći:

Izračunavanje uvjeta koji je pridružen petlji "*dok je*" ("*while*"). Rezultat izračuna uvjeta je uvijek ili istina ili neistina. Uvjet je potrebno definirati pomoću varijabli čije se vrijednosti mijenjaju u tijelu petlje tijekom izvođenja petlje kako bi onda mogla završiti. Za uređaje koji rade neprekidno 24 sata na dan i 7 dana u tjednu (označava se i kao 24/7) kao uvjet se navodi vrijednost logičko 1. Time nastaje tzv. neprekinuta (beskonačna) petlja koja nikada ne završava sa svojim izvođenjem. Primjer takvih uređaja su pametna osjetila, upravljačka računala za semaforizirana raskršća, web poslužitelji i sl.;

Uvjet ispunjen (istinit) karakterizira slučaj kada se izvršava tijelo petlje, odnosno sve naredbe u tijelu petlje. Nakon što se izvrše sve naredbe u tijelu petlje kao sljedeća naredba ponovno se izvodi izračunavanje uvjeta petlje "*dok je*" ("*while*");

Uvjet nije ispunjen (neistinit) karakterizira slučaj završetka petlje. Program tada izlazi iz petlje i kao sljedeća naredba izvrši se prva naredba nakon cijele strukture petlje "*dok je*" ("*while*").

Tijek petlje "*dok je*" ("*while*") može se promotriti pomoću pseudokôda 6.12. Vidljivo je da se program izvodi naredbu po naredbu slijedno dok ne dođe do početka petlje "*dok je*" ("*while*"). Uvjet se ispita i ako je ispunjen (istinit) izvedu se naredbe u tijelu petlje. Nakon toga ponovno se ispita uvjet. Ovo je glavna razlika u odnosu na strukturu grananja, gdje se izvršavanje programa nastavlja s prvom sljedećom naredbom nakon strukture grananja. Ako je uvjet i dalje ispunjen, ponovno se izvršavaju naredbe tijela petlje. Nakon izvršavanja tijela petlje ispituje se uvjet i ova se procedura ponavlja tako dugo dok ne nastupi slučaj da uvjet petlje više nije ispunjen. Time petlja završava i kao sljedeća naredba se izvrši prva naredba nakon cijele strukture petlje "*dok je*" ("*while*"). Moguće je primijetiti da se nakon izvršavanja tijela petlje uvijek ponovno ispitao uvjet, a program nastavlja s izvršavanjem naredbi koje dolaze nakon cijele strukture petlje "*dok je*" ("*while*") tek kada nastupi slučaj da uvjet više nije ispunjen. Kraj strukture petlje "*dok je*" ("*while*") se označava ključnom riječi "*kraj dok je*".

Pseudokôd 6.12: Petlja "*dok je*" ("*while*").

```
naredbe prije petlje
dok je uvjet činiti
   naredbe tijela petlje
kraj dok je
naredbe nakon petlje
```

Primjena petlje "*dok je*" ("*while*") može se dobro objasniti na jednostavnom primjeru izračuna zbroja prvih "*n*" cijelih pozitivnih brojeva. Rješenje ovog problema dano je pseudokôdom 6.13. Za objašnjenje rješenja ovog problema potrebno je uzeti u obzir način rada računala, odnosno objašnjenje dano uz naredbu 3.3. Matematički se traženi zbroj može napisati u obliku:

$$zbroj = \sum_{i=1}^{n} i = 1 + 2 + 3 + ... + n, \tag{6.1}$$

gdje *zbroj* označava traženi rezultat, i označava trenutnu vrijednost koja se pribraja i n gornju granicu do koje će se i povećati. Dalje se izračun traženog

zbroja može raspisati za pojedine početne slučajeve kao:

$$zbroj(1) = \sum_{i=1}^{1} i = 1 = 1, \tag{6.2}$$

$$zbroj(2) = \sum_{i=1}^{2} i = 1 + 2 = zbroj(1) + 2 = 3, \tag{6.3}$$

$$zbroj(3) = \sum_{i=1}^{3} i = 1 + 2 + 3 = zbroj(2) + 3 = 6. \tag{6.4}$$

Iz izraza 6.2 do 6.4 vidljivo je kako je moguće koristiti međurezultat zbroja, odnosno zbroj prvih n cijelih brojeva jednak je zbroju prvih $n-1$ cijelih brojeva i broja n. Ta zakonitost omogućuje korištenje konstrukcije naredbe objašnjene izrazom 3.3 u sklopu petlje. Ova zakonitost se posebno može iskoristiti jer u petlji trenutni iznos zbroja predstavlja međurezultat za sljedeću iteraciju izvršavanja petlje. Jedino je bitno dobro definirati početnu (inicijalnu) vrijednost varijable u koju će se spremiti vrijednost zbroja. Potrebno je odabrati vrijednost koja neće utjecati na iznos zbroja, a to je vrijednost 0.

Pseudokôd 6.13: Izračun zbroja prvih "n" cijelih brojeva.

```
Ulaz:
Izlaz:
Deklaracija:
        brojPodataka, zbroj, brojiloPodataka
Inicijalizacija:
        zbroj := 0
        brojiloPodataka := 1
Ispis:
        "Unesite količinu podataka >"
Unos:
        brojPodataka
dok je brojiloPodataka <= brojPodataka činiti
   zbroj := zbroj + brojiloPodataka
   brojiloPodataka := brojiloPodataka + 1
kraj dok je
Ispis:
        "Zbroj prvih " + brojPodataka + " cijelih brojeva iznosi " + zbroj
```

U pseudokôdu 6.13 koriste se tri varijable. Pri tome se varijabla "*brojPodataka*" koristi za spremanje količine brojeva koju je potrebno zbrojiti, varijabla "*zbroj*" služi za spremanje rezultata dok varijabla "*brojiloPodataka*" predstavlja kontrolnu varijablu potrebnu da se prati količina brojeva koja je već obrađena, odnosno pridodana varijabli "*zbroj*". Prikazano rješenje se izvodi tako da se u prvom dijelu deklariraju potrebne varijable, a zatim im se pridružuju početne (inicijalne) vrijednosti. Varijabli "*zbroj*" pridružuje se početna vrijednost 0 iz opisanih razloga, a varijabli "*brojiloPodataka*" pridružuje se početna vrijednost 1. Zatim se radi unos količine podataka za obradu (u ovom slučaju zbrajanje) uz ispis odgovarajuće poruke operateru.

Nakon što je operater unio traženu količinu podataka za obradu, na redu je petlja. Uvjet petlje je "$brojiloPodataka <= brojPodataka$" i taj uvjet će osigurati izvršavanje tijela petlje zadnji puta kad je varijabla "*brojiloPodataka*" identična varijabli "*brojPodataka*". Pri tome je potrebno u tijelo petlje uključiti naredbu koja pri svakom izvršavanju tijela petlje vrijednost varijable "*brojiloPodataka*" uveća za vrijednost 1. Tako će se tijelo petlje izvršiti točno onoliko puta koliko je unesena vrijednost varijable "*brojPodataka*" i kao rezultat rada petlje dobit će se zbroj prvih cijelih "*n*" brojeva. U slučaju da se u tijelo petlje ne uključi naredba koja mijenja jednu od varijabli koje se ispituju u uvjetu, dobiva se beskonačna petlja. Varijabli "*brojiloPodataka*" moguće je zadati i drugu početnu vrijednost pri čemu je bitno prilagoditi uvjet petlje. Npr. za početnu vrijednost 0 potrebno je uvjet petlje promijeniti u "$brojiloPodataka < brojPodataka$". Tako će se tijelo petlje opet izvršiti onoliko puta koliko je unesena vrijednost varijable "*brojPodataka*". Pri tome je potrebno prilagoditi i tijelo petlje na način da se zamijeni redoslijed naredbi za osvježavanje vrijednosti varijabli "*zbroj*" i "*brojiloPodataka*". Razlog tome je da se varijabli "*zbroj*" prvo doda vrijednost 1, a ne 0 koliko iznosi početna vrijednost varijable "*brojiloPodataka*" u ovom slučaju. Na kraju programskog rješenja ispisuje se dobiveni rezultat zbroja u pripadajućoj poruci za operatera.

6.5.2 Petlja "*ponavljati - do*" ("*do - while*")

Petlja "*ponavljati - do*" ("*do - while*") ispituje uvjet nakon što se izvrši tijelo petlje. To znači da će se tijelo petlje izvršiti najmanje jednom. Formalno zapisano tijek petlje "*ponavljati - do*" ("*do - while*") je sljedeći:

Izvršavanje tijela petlje koje je pridruženo petlji "*ponavljati - do*" ("*do - while*"). Tijelo petlje se izvršava prvi puta bez da je provjeren uvjet petlje "*ponavljati - do*" ("*do - while*"). Izvrše se sve naredbe tijela petlje pri čemu je opet u tijelo petlje potrebno uključiti naredbu koja će mijenjati vrijednost jedne od varijabli (kontrolnu varijablu) koja se provjerava u uvjetu kako

bi se izbjegla beskonačna petlja;

Uvjet istinit karakterizira slučaj kada se ponovno izvršava tijelo petlje, odnosno sve naredbe u njemu. Nakon što se izvrše sve naredbe u tijelu petlje kao sljedeća naredba ponovno se izračunava uvjet, odnosno provjerava se uvjet petlje "*ponavljati - do*" ("*do - while*");

Uvjet neistinit karakterizira slučaj završetka petlje. Program tada izlazi iz petlje i kao sljedeća naredba se izvrši prva naredba nakon cijele strukture petlje "*ponavljati - do*" ("*do - while*").

Pseudokôd 6.14: Petlja "*ponavljati - do*" ("*do - while*").

```
naredbe prije petlje
ponavljati
  naredbe tijela petlje
do uvjet
naredbe nakon petlje
```

Tijek petlje "*ponavljati - do*" ("*do - while*") može se promotriti pomoću pseudokôda 6.14. Vidljivo je kako se program izvodi klasično naredbu po naredbu dok ne dođe do početka petlje "*ponavljati - do*" ("*do - while*"). Za razliku od petlje "*dok je*" ("*while*") prvo se izvedu sve naredbe u tijelu petlje. Tek nakon izvršavanja tijela petlje ispituje se uvjet. Ako je on ispunjen, ponovno se izvodi tijelo petlje. Ova se procedura ponavlja tako dugo dok se ne pojavi slučaj da uvjet petlje više nije ispunjen. Time petlja završava i kao sljedeća naredba se izvrši prva naredba nakon cijele strukture petlje "*ponavljati - do*" ("*do - while*"). Moguće je primijetiti kako se nakon izvršavanja tijela petlje uvijek ponovno ispituje uvjet, a program nastavlja s izvršavanjem naredbi koje dolaze nakon cijele strukture petlje "*ponavljati - do*" ("*do - while*") tek u slučaju da uvjet nije ispunjen.

Primjena petlje "*ponavljati - do*" ("*do - while*") može se objasniti na jednostavnom primjeru računanja umnoška prvih "n" cijelih pozitivnih brojeva. U matematici se taj izračun naziva faktorijeli. Rješenje ovog problema prikazano je pseudokôdom 6.15. Za objašnjenje rješenja ovog problema potrebno je ponovno uzeti u obzir način rada računala, odnosno objašnjenje dano uz naredbu 3.3. Matematički se traženi umnožak može napisati u obliku:

$$umnožak = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdot ... \cdot n, \tag{6.5}$$

gdje *umnožak* označava traženi rezultat, i označava trenutnu vrijednost kojom se množi, a n gornju granicu do koje će se i povećavati. Dalje se izračun traženog umnoška može raspisati za pojedine početne slučajeve kao:

$$umnožak(1) = \prod_{i=1}^{1} i = 1 = 1, \tag{6.6}$$

$$umnožak(2) = \prod_{i=1}^{2} i = 1 \cdot 2 = umnožak(1) \cdot 2 = 2, \tag{6.7}$$

$$umnožak(3) = \prod_{i=1}^{3} i = 1 \cdot 2 \cdot 3 = umnožak(2) \cdot 3 = 6. \tag{6.8}$$

Iz izraza 6.6 do 6.8 ponovno je vidljiva mogućnost korištenja međurezultata umnoška, odnosno umnožak prvih n cijelih brojeva jednak je umnošku prvih $n-1$ cijelih brojeva i broja n. Ta zakonitost omogućuje korištenje konstrukcije naredbe objašnjene izrazom 3.3 u sklopu petlje. U ovom slučaju u tijelu petlje trenutni iznos umnoška predstavlja međurezultat za sljedeću iteraciju izvršavanja petlje. Bitno je dobro definirati početnu (inicijalnu) vrijednost varijable u koju će se spremiti vrijednost umnoška i odabrati vrijednost koja neće utjecati na iznos umnoška, a to je u ovom slučaju vrijednost 1.

U pseudokôdu se koriste tri varijable. Pri tome varijabla "*brojPodataka*" služi za spremanje količine brojeva koju je potrebno obraditi. Varijabla "*umnozak*" služi za spremanje rezultata dok varijabla "*brojiloPodataka*" predstavlja kontrolnu varijablu za praćenje količine obrađenih podataka. Prikazano rješenje se izvodi tako da se u prvom dijelu prvo deklariraju potrebne varijable, a zatim im se pridružuju početne vrijednosti. Varijabli "*umnozak*" pridružuje se vrijednost 1 iz opisanih razloga, a varijabli "*brojiloPodataka*" pridružuje se početna vrijednost 1 kako bi se osigurala ispravna vrijednost za početak množenja u prvoj iteraciji petlje. Nakon toga se radi unos količine podataka za obradu (u ovom slučaju množenje) uz ispis odgovarajuće poruke operateru.

Nakon što je operater unio traženu količinu podataka za obradu, na redu je petlja. Za razliku od prethodne vrste petlje, sada se prvo izvodi tijelo petlje. Potrebno je ovdje primijetiti kako uvjet petlje još nije ispitan, a tijelo petlje se izvršilo u prvoj iteraciji. Tek sada se ispituje uvjet petlje. U ovom slučaju uvjet ponovno glasi "*brojiloPodataka <= brojPodataka*" i osigurava da će se tijelo petlje izvršiti zadnji put kad je varijabla "*brojiloPodataka*" identična varijabli "*brojPodataka*". Pri tome je ponovno u tijelo petlje uključena naredba koja u svakoj iteraciji petlje uveća varijablu "*brojiloPodataka*" za 1. Tako će se tijelo petlje izvršiti točno onoliko puta koliko je unesena vrijednost varijable "*brojPo-*

Pseudokôd 6.15: Izračun umnoška prvih "n" cijelih pozitivnih brojeva.

```
Ulaz:
Izlaz:
Deklaracija:
        brojPodataka, umnozak, brojiloPodataka
Inicijalizacija:
        umnozak := 1
        brojiloPodataka := 1
Ispis:
        "Unesite količinu podataka >"
Unos:
        brojPodataka
ponavljati
   umnozak := umnozak * brojiloPodataka
   brojiloPodataka := brojiloPodataka + 1
do brojiloPodataka <= brojPodataka
Ispis:
        "Umnožak prvih " + brojPodataka + " cijelih brojeva iznosi "
        + umnozak
```

dataka". Kao rezultat izvršavanja petlje u varijabli "*umnozak*" bit će spremljena vrijednost umnoška prvih "n" cijelih brojeva. Na kraju programskog rješenja se ispisuje dobiveni rezultat umnoška u pripadajućoj poruci za operatera.

6.5.3 Petlja *"za - do"* ("*for*")

Prilikom izrade programskog kôda za petlje *"dok je"* ("*while*") i *"ponavljati - do"* ("*do - while*") bitno je prije petlje napraviti potrebne inicijalizacije varijabli i u tijelo petlje uključiti naredbu koja mijenja vrijednost kontrolne varijable. Takav pristup je dobar u slučaju kada broj izvršavanja tijela petlje (iteracija) nije unaprijed poznat. U slučaju kada je broj izvršavanja unaprijed poznat mnogo je bolje iskoristiti petlju *"za - do"* ("*for*") koja u svojoj definiciji uključuje inicijalizaciju početne vrijednosti kontrolne varijable, definiranje uvjeta petlje te promjene kontrolne varijable. Kontrolna varijabla se najčešće mijenja za 1 pri svakoj iteraciji pa se takva petlja *"za - do"* ("*for*") može definirati pseudokôdom 6.16. U slučaju da se kontrolna varijabla mijenja u svakoj iteraciji za neku drugu vrijednost koristi se definicija petlje *"za - do"* ("*for*") pomoću pseudokôda 6.17. Pri tome se vrijednost promjene kontrolne varijable označava kao korak "k". Kraj strukture petlje "*za - do*" ("*for*") označava ključna riječ "*kraj za*".

U oba pseudokôda se prije strukture petlje nalaze naredbe koje služe za

Pseudokôd 6.16: Petlja *"za - do"* ("*for*") uz promjenu kontrolne varijable za 1.

naredbe prije petlje
za $i := početnaVrijednost$ **do** $konačnaVrijednost$ **činiti**
  naredbe tijela petlje
kraj za
naredbe nakon petlje

Pseudokôd 6.17: Petlja *"za - do"* ("*for*") uz definiciju koraka promjene kontrolne varijable.

naredbe prije petlje
za $i := početnaVrijednost$ **do** $konačnaVrijednost$ **korak** k **činiti**
  naredbe tijela petlje
kraj za
naredbe nakon petlje

deklaraciju, inicijalizaciju i unos varijabli te druge obrade podataka vezane za izvođenje programa. U liniji pseudokôda za oznaku početka petlje *"za - do"* ("*for*") vidljivo je navođenje imena varijable "i", koja služi kao kontrolna varijabla te njene inicijalizacije. Navodi se i njena konačna vrijednost što znači da se petlja *"za - do"* ("*for*") izvodi dok kontrolna vrijednost ne dostigne definiranu konačnu vrijednost. Pri tome se kontrolna varijabla mijenja na kraju svake iteracije za vrijednost 1 u slučaju pseudokôda 6.16, odnosno za vrijednost koraka "k" u slučaju pseudokôda 6.17. Ovisno o tome je li konačna vrijednost veća ili manja od početne vrijednost radi se uvećavanje (inkrement) ili smanjivanje (dekrement) kontrolne varijable. Formalno zapisano tijek petlje *"za - do"* ("*for*") je sljedeći:

Inicijalizacija se obavlja samo jednom na početku petlje *"za - do"* ("*for*"). U ovom dijelu se postavlja početna vrijednost kontrolne varijable, a po potrebi se može izvršiti i deklaracija kontrolne varijable;

Uvjet istinit karakterizira slučaj kada se izvršava tijelo petlje, odnosno sve naredbe u njemu. U uvjetu se ispituje je li kontrolna varijabla dostigla zadanu konačnu vrijednost;

Uvjet neistinit karakterizira slučaj završetka petlje. Program tada izlazi iz petlje i kao sljedeća naredba se izvrši prva naredba nakon cijele strukture petlje *"za - do"* ("*for*");

Izvršavanje tijela petlje koje je pridruženo petlji *"za - do"* ("*for*"). Nakon što se

Pseudokôd 6.18: Izračun zbroja cijelih brojeva unutar intervala.

Ulaz:
Izlaz:
Deklaracija:
    i, zbroj, donjaGranica, gornjaGranica
Inicijalizacija:
    zbroj := 0
Ispis:
    "Unesite vrijednost donje granice intervala >"
Unos:
    donjaGranica
Ispis:
    "Unesite vrijednost gornje granice intervala >"
Unos:
    gornjaGranica
za $i := donjaGranica$ **do** $gornjaGranica$ **činiti**
  zbroj := zbroj + i
kraj za
Ispis:
    "Zbroj cijelih brojeva unutar intervala iznosi " + zbroj

izvrše sve naredbe u tijelu petlje, kao sljedeća naredba se izvodi naredba za promjenu vrijednosti kontrolne varijable zadana u definiciji petlje *"za - do"* ("*for*"). Nakon toga se petlja nastavlja provjerom uvjeta te izvršavanjem tijela petlja ako je uvjet ispunjen.

Primjena petlje *"za - do"* ("*for*") može se dobro objasniti na primjeru izračuna zbroja cijelih brojeva koji se nalaze unutar intervala zadanog gornjom i donjom granicom. Programsko rješenje uz korištene petlje *"za - do"* ("*for*") dano je pseudokôdom 6.18. U rješenju je pretpostavljeno da su donja i gornja granica intervala uključene. U pseudokôdu 6.18 su definirane četiri varijable. Varijabla "*i*" predstavlja kontrolnu varijablu za praćenje količine obrađenih podataka. Varijabla "*zbroj*" služi za spremanje rezultata rada programa dok varijable "*donjaGranica*" i "*gornjaGranica*" služe za spremanje granica intervala.

U prvom dijelu pseudokôda 6.18 deklarirane su potrebne varijable, a zatim se unose potrebne vrijednosti za pojedine varijable. Potom dolazi dio petlje *"za - do"* ("*for*"). Kako je zadatak izračunati zbroj cijelih brojeva unutar intervala s uključenim granicama, kao početna vrijednost kontrolne varijable "*i*" uzima se vrijednost varijable "*donjaGranica*". Kao konačna vrijednost, koju kontrolna varijabla "*i*" ima dostići uzima se vrijednost varijable "*gornjaGranica*". Vrije-

dnost promjene kontrolne varijable je u ovom slučaju 1 jer je potrebno zbrojiti cijele brojeve. Iz tog razloga u definiciji petlje *"za - do"* ("*for*") nije potrebno zadavati vrijednost koraka pa se koristi oblik petlje *"za - do"* ("*for*") dan pseudokôdom 6.16. Tijelo petlje sa sastoji samo od naredbe za promjenu vrijednosti varijable "*zbroj*" jer promjenu kontrolne varijable izvršava petlja *"za - do"* ("*for*") automatski. Nije potrebno da ju programer posebno definira i to predstavlja prednost prema petljama *"dok je"* ("*while*") i *"ponavljati - do"* ("*do - while*"). Naredba vezana za promjenu kontrolne varijable izvrši se nakon tijela petlje i u slučaju ovog programskog rješenje ona glasi "*i := i +1*". Tek nakon što se izvrši naredba za promjenu kontrolne varijable ispituje se uvjet. Kada uvjet više nije ispunjen, petlja *"za - do"* ("*for*") završava s radom i izvodi se naredba za ispis rezultata izvršavanja programa koja se nalazi iza cijele strukture petlje *"za - do"* ("*for*").

6.6 Primjeri

U ovom potpoglavlju dano je nekoliko primjera koji rješavaju neke od najčešćih problema za implementaciju programa za upravljanje nekim iterativnim postupkom korištenjem gore objašnjenih struktura grananja i petlji. Rješenje svakog primjera sastoji se od kratkog objašnjenja ideje rješenja, popisa varijabli, pseudokôda te objašnjenja pseudokôda. Primjeri u sljedeća dva poglavlja bit će vezani za primjere dane u nastavku ovog poglavlja. Tako popis varijabli i detaljno objašnjenje rada više neće biti ponovljeni jer će se rješenje dano u obliku dijagrama toka ili programskog jezika C♯ zasnivati na u ovom poglavlju objašnjenom rješenju. Naznačiti će se samo specifičnosti rješenja danog pomoću dijagrama toka ili u programskom jeziku C♯.

■ **Primjer 6.1**

Potrebno je izraditi pseudokôd programa koji će ispitati je li se uneseni cijeli broj nalazi u zadanom intervalu. Rezultat je potrebno ispisati u odgovarajućoj poruci operateru. Pri tome je donja granica intervala uključena, a gornja granica nije uključena u interval.

Rješenje:

Kao prvi korak u izradi rješenja definirati će se potrebne varijable. Potrebne su tri varijable: (i) varijabla za spremanje podatka za obradu; (ii) varijabla za spremanje donje granice intervala; i (iii) varijabla za spremanje gornje granice intervala. Kako se obrađuju cijeli brojevi, sve varijable mogu biti istog tipa,

Ime varijable	Tip varijable	Značenje varijable
podatak	cijeli broj	Podatak za obradu
donjaGranica	cijeli broj	Donja granica zadanog intervala
gornjaGranica	cijeli broj	Gornja granica zadanog intervala

Tablica 6.2: Popis varijabli za kreiranje idejnog programskog rješenja provjere pripadnosti intervalu.

odnosno cijeli broj. Popis potrebnih varijabli je dan u tablici 6.2.

Nakon što su definirane potrebne varijable, potrebno je definirati uvjet za provjeru pripadnosti intervalu. Prema tablici 6.2 interval je definiran donjom i gornjom granicom. U tekstu primjera naznačeno je da je donja granica uključena u interval dok gornja granica nije uključena. Da bi neki podatak pripadao nekom intervalu potrebno je ispuniti dva uvjeta. Potrebno je da podatak bude jednak ili veći od donje granice i istovremeno manji od gornje granice. Donja granica je uključena u interval vrijednosti tako da je to potrebno uključiti u uvjet. Spajanje navedena dva uvjeta, odnosno dvije relacije provjere, potrebno je napraviti korištenjem logičke funkcije. Kako je potrebno ispuniti oba uvjeta, iskoristiti će se logička funkcija I. Konačno se uvjet potreban za provjeru pripadnosti intervalu može napisati kao:

$$(podatak >= donjaGranica)\ I\ (podatak < gornjaGranica). \qquad (6.9)$$

Ako je logički izraz 6.9 ispunjen, uneseni podatak pripada intervalu, a u slučaju da nije ispunjen uneseni podatak ne pripada intervalu. Time su dobivena dva različita slučaja za definiranje grananja i dovoljno je u svakom slučaju ispisati pripadnu poruku operateru. Rješenje se sada može napisati u obliku pseudokôda 6.19. U prvom dijelu se deklariraju potrebne varijable, a zatim se radi unos podataka. Nakon unosa vrši se provjera korištenjem strukture grananja "*if - else*" uz uvjet dan izrazom 6.9. Ovisno o ispunjenosti uvjeta ispisuje se pripadna poruka operateru koja označava pripadnost unesenog podatka zadanom intervalu. Nakon što se izvrši struktura grananja, program završava s radom.

■ Primjer 6.2

Potrebno je napraviti prijedlog programskog rješenja za izračun cestarine za sve klase vozila korištenjem skretnice. Pretpostavite kako se radi o jednostavnoj dionici autoceste koja ima samo jedan ulaz i jedan izlaz. Također je potrebno izračunati iznos PDV-a te ga ispisati u odgovarajućoj poruci operateru.

Pseudokôd 6.19: Provjera pripadnosti intervalu.

Ulaz:
Izlaz:
Deklaracija:
    podatak, donjaGranica, gornjaGranica
Ispis:
    "Unesite vrijednost podatka za provjeru pripadnosti >"
Unos:
    podatak
Ispis:
    "Unesite vrijednost donje granice intervala >"
Unos:
    donjaGranica
Ispis:
    "Unesite vrijednost gornje granice intervala >"
Unos:
    gornjaGranica
ako je (podatak >= donjaGranica) I (podatak < gornjaGranica) **onda**
  Ispis:
    "Unesena vrijednost pripada intervalu."
inače
  Ispis:
    "Unesena vrijednost ne pripada intervalu."
kraj ako

Rješenje:

Kako bi se općenito mogla izračunati cijena cestarine potrebno je poznavati: (i) postotak PDV-a koji se obračunava; (ii) kategoriju vozila; (iii) mjesta ulaska na autocestu; te (iv) mjesto izlaska s autoceste. U ovom primjeru se pretpostavlja da se cestarina obračunava za jednostavnu autocestu sa samo jednim ulazom i jednim izlazom. Stoga je za izračun cestarine dovoljno poznavati kategoriju vozila te postotak PDV-a. Osim ove dvije varijable, potrebno je još definirati varijable za spremanje iznosa cestarine bez PDV-a te spremanje iznosa PDV-a kojeg je potrebno platiti. Konačni popis varijabli prikazan je u tablici 6.3. Varijable u koje se spremaju konačni iznosi PDV-a te cestarine su tipa broj s plivajućim zarezom (realni brojevi), a ostale varijable su tipa cijeli broj. Tip cijeli broj pogodan je za spremanje kategorije vozila za naplatu cestarine jer u Republici Hrvatskoj postoji 5 različitih kategorija cestovnih vozila za naplatu cestarine.

Ime varijable	Tip varijable	Značenje varijable
vozilo	cijeli broj	Oznaka kategorije vozila
cijena	broj s plivajućim zarezom	Iznos cestarine bez PDV-a
PDV	cijeli broj	Iznos PDV-a u postocima
iznosPDVa	broj s plivajućim zarezom	Konačni iznos PDV-a

Tablica 6.3: Popis varijabli idejnog programskog rješenja za izračun cestarine.

Predloženo rješenje prikazano je pseudokôdom 6.20 i 6.21. Sastoji se u prvom dijelu od deklaracije potrebnih varijabli te unosa vrijednosti. Pri tome se može primijetiti kako je prilikom unosa podatka o kategoriji vozila u pripadnoj poruci za operatera ispisana i detaljnija pomoć. Ona sadrži ispis zakonskih oznaka pojedinih kategorija vozila i u zagradi numeričku oznaku koja se koristi u programu. Nakon unosa podataka počinje skretnica koja je vezana za vrijednost spremljenu u varijabli "*vozilo*". Ovisno o spremljenoj vrijednosti aktivira se pripadni slučaj u kojem se varijabli "*cijena*" pridružuje osnovna cijena cestarine bez PDV-a. U slučaju da nije definirana ispravna kategorija aktivira se slučaj "*zadano*" koji varijabli "*cijena*" pridružuje vrijednost $-1,0$. Pomoću te vrijednosti moguće je detektirati pogrešan unos kategorije vozila jer je svaka regularna cijena veća od nule. Nakon skretnice nalazi se grananje "*if - else*" koje provjera ispravnost određivanja cijene cestarine bez PDV-a. Kao uvjet provjerava se je li vrijednost varijable "*cijena*" veća od nule. Ako jest, izračunava se iznos PDV-a i kreće ispis rezultata. Potrebno je primijetiti kako se, kod ispisa ukupnog iznosa cestarine s PDV-om, prilikom ispisa izračunava ukupan iznos cestarine. To se izvršava pomoću izraza "$cijena + iznosPDVa$", koji je smješten u okrugle zagrade. One su potrebne jer prilikom ispisa operator "+" označava spajanje znakovnih nizova, a za izračun cijene cestarine je potreban aritmetički operator zbrajanja. U slučaju da unos kategorije vozila nije bio ispravan, operateru se ispisuje poruka o grešci.

■ **Primjer 6.3**

Mnogokut (poligon) je dio ravnine omeđen zatvorenom izlomljenom linijom ili manje formalno dio ravnine omeđen ravnim dužinama koje možemo nacrtati, a da ne podižemo olovku. Opseg mnogokuta jednak je zbroju duljina svih njegovih stranica. Potrebno je napraviti program koji će izračunati opseg mnogokuta korištenjem strukture petlje "while".

Rješenje:

Prema općenitoj matematičkoj definiciji opseg je duljina zatvorene krivulje.

Pseudokôd 6.20: Izračun cijene cestarine.

```
Ulaz:
Izlaz:
Deklaracija:
        vozilo, cijena, PDV, iznosPDVa
Ispis:
        "Unesite postotak PDV-a:"
Unos:
        PDV
Ispis:
        "Unesite kategoriju cestovnog vozila:"
                "Kategorija vozila IA (1)"
                "Kategorija vozila I (2)"
                "Kategorija vozila II (3)"
                "Kategorija vozila III (4)"
                "Kategorija vozila IV (5)"
Unos:
        vozilo
skretnica (vozilo)
        slučaj 1:
                cijena := 10.0
        slučaj 2:
                cijena := 20.0
        slučaj 3:
                cijena := 30.0
        slučaj 4:
                cijena := 40.0
        slučaj 5:
                cijena := 50.0
        zadano:
                cijena := -1
kraj skretnica
```

To znači da je u slučaju mnogokuta potrebno zbrojiti duljine pojedinih stranica mnogokuta kako bi se izračunao njegov opseg. U slučaju općenitog mnogokuta nije unaprijed poznat broj stranica kao u slučaju trokuta ili pravokutnika pa je potrebno iskoristiti strukturu petlje u rješavanju ovog problema. Od varijabli potrebno je spremiti boj kutova mnogokuta, duljinu pojedine stranice, opseg te broj učitanih duljina stranica. Navedene varijable su dane u tablici 6.4.

Skica programa za izračun opsega mnogokuta dana je pseudokôdom 6.22. U

Pseudokôd 6.21: Izračun cijene cestarine (nastavak).

```
ako je cijena > 0 onda
  Izračunaj:
        iznosPDVa := cijena * PDV / 100
  Ispis:
        "Cijena cestarine bez PDV-a iznosi " + cijena + " HRK"
        "PDV iznosi " + iznosPDVa + " HRK"
        "Cestarina iznosi " + (cijena + iznosPDVa) + " HRK"
inače
  Ispis:
        "GREŠKA! Cijena cestarine nije mogla biti izračunata!"
kraj ako
```

Ime varijable	Tip varijable	Značenje varijable
brojKutova	cijeli broj	Broj kutova mnogokuta
stranica	broj s plivajućim zarezom	Duljina jedne stranice mnogokuta
opseg	broj s plivajućim zarezom	Opseg mnogokuta
i	cijeli broj	Kontrolna varijabla petlje "*while*"

Tablica 6.4: Popis varijabli idejnog programskog rješenja za izračun opsega mnogokuta.

prvom dijelu pseudokôda deklariraju se i inicijaliziraju varijable. Zatim slijedi unos broja kutova u mnogokutu kako bi se dohvatio broj potrebnih iteracija tijela petlje "*while*". Nakon unosa započinje petlja. Kontrolna varijabla "*i*" uspoređuje se s varijablom "*brojKutova*" i dok god je kontrolna varijabla manja od broja kutova mnogokuta potrebno je izvršiti tijelo petlje "*while*". U tijelu petlje prvo se unosi duljina pojedine stranice mnogokuta koja se zatim pridodaje trenutnom iznosu opsega. Zatim se povećava kontrolna varijabla "*i*" te se ponovno provjerava uvjet petlje "*while*". Prilikom ispisa poruke operateru za unos duljine stranice u okruglim je zagradama navedena naredba "*i + 1*". Ona je potrebna kako bi se prilikom unosa duljine stranice prvo zatražila duljina prve stranice jer je početna vrijednost kontrolne varijable 0. Oble zagrade su potrebne kako bi se prvo izvršila aritmetička operacija zbrajanja, a tek nakon toga operacija spajanja znakovnih nizova za ispis. Bez oblih zagrada, program bi vrijednost 1 shvatio kao niz znakova koji je potrebno ispisati na zaslonu računala i poruka operateru ne bi bila ispravna. Nakon tijela petlje radi se ispis izračunatoga opsega mnogokuta na zaslon računala.

■ **Primjer 6.4**

Pseudokôd 6.22: Izračun opsega mnogokuta.

```
Ulaz:
Izlaz:
Deklaracija:
        brojKutova, stranica, opseg, i
Inicijalizacija:
        i := 0
        opseg := 0
Ispis:
        "Unesite broj kutova u mnogokutu >"
Unos:
        brojKutova
dok je i < brojKutova činiti
  Ispis:
          "Unesite duljinu " + (i + 1) + ". stranice mnogokuta >"
  Unos:
          stranica
  opseg := opseg + stranica
  i := i + 1
kraj dok je
Ispis:
        "Opseg mnogokuta iznosi " + opseg
```

Operacija dijeljenja jedna je od aritmetičkih operacija koja nije definirana za sve moguće vrijednosti djelitelja. Naime, dijeljenje s nulom nije definirano, što znači da je operaciju dijeljenja moguće izvršiti samo u slučaju ako je djelitelj različit od nule. Potrebno je izraditi program koji će imati ugrađenu zaštitu kod unosa podataka zasnovanu na petlji "do - while".

Rješenje:

U dosadašnjim programima se prilikom unosa traženih vrijednosti nije radila provjera. Pretpostavka da će operater svaki puta unijeti ispravan podatak nije dobra pri svakodnevnom korištenju računala. Iz tog razloga se prilikom unosa podataka radi provjera njihove ispravnosti. Za većinu podataka je moguće napraviti osnovnu provjeru ispravnosti. Tako masa vozila može samo biti veća od nule, broj slobodnih parkirnih mjesta može također biti samo veći ili jednak nuli i sl. Jednako je tako potrebno kod matematičkih operacija provjeriti njihove iznose prije samog izračuna. Tako se može izračunati korijen samo pozitivnog broja, arkus sinus se može izračunati samo za vrijednosti iz intervala $[-1, 1]$ i sl.

U slučaju dijeljenja vrijedi pravilo da nije moguće dijeliti s nulom. To znači da djelitelj mora biti različit od nule dok djeljenik može imati bilo koju vrijednost. Prilikom unosa podataka je zato potrebno provjeriti vrijednost djelitelja koju je operator unio. Za tu se svrhu može primijeniti petlja "*do - while*". U njenom tijelu je potrebno staviti naredbe za unos željenog podatka dok uvjet petlje sadrži provjeru ispravnosti unesenog podatka. Za dijeljenje dva broja potrebno je deklarirati ukupno tri varijable kako bi se mogao spremiti djelitelj, djeljenik i količnik. Popis varijabli za program koji rješava ovaj problem dan je u tablici 6.5.

Ime varijable	Tip varijable	Značenje varijable
kolicnik	broj s plivajućim zarezom	Rezultat dijeljenja (količnik)
djeljenik	broj s plivajućim zarezom	Broj koji se dijeli
djelitelj	broj s plivajućim zarezom	Broj kojim se dijeli

Tablica 6.5: Popis varijabli idejnog programskog rješenja za zaštitu unosa podataka kod aritmetičke operacije dijeljenja.

Skica programa za rješenje ovog problema dana je pseudokôdom 6.23. U prvom dijelu deklariraju se sve potrebne varijable. Zatim se unosi djeljenik za koji nije potrebno napraviti provjeru. Sljedeći unos se odnosi na djelitelj koji ne smije biti jednak nuli tako da je uneseni iznos sada potrebno provjeriti. Za to je iskorištena petlja "*do - while*". U njenom tijelu su naredbe za unos vrijednosti varijable "*djelitelj*" dok uvjet petlje provjerava ispravnost unesene vrijednosti. U slučaju da je unesena vrijednost neispravna (u ovom slučaju identična nuli) potrebno je ponovno izvršiti unos. Petlja će se prekinuti kada je unesena vrijednost ispravna, odnosno različita od nule. Nakon petlje izračunava se rezultat dijeljenja i ispisuje se rezultat u pripadnoj poruci operateru.

■ **Primjer 6.5**

Student Fakulteta prometnih znanosti želi se natjecati za stručnu praksu u inozemstvu. Za ispunjavanje prijavnice potreban mu je prosjek ocjena iz svih do sada položenih ispita. Kako bi se studentu olakšao izračun prosječne ocjene potrebno je napraviti program koji će srednju ocjenu do sada položenih predmeta izračunati korištenjem petlje "for".

Rješenje:

Općenito se prosječna ocjena može izračunati kao omjer zbroja svih ocjena i broja ocjena. Matematički se to može izraziti kao:

Pseudokôd 6.23: Dijeljenje dva broja uz zaštitu za unos ispravnih podataka.

```
Ulaz:
Izlaz:
Deklaracija:
        kolicnik, djeljenik, djelitelj
Ispis:
        "Unesite djeljenik >"
Unos:
        djeljenik
ponavljati
  Ispis:
          "Unesite djelitelj različit od nule >"
  Unos:
          djelitelj
do djelitelj = 0
Izračunaj:
        kolicnik := djeljenik / djelitelj
Ispis:
        "Rezultat dijeljenja iznosi " + kolicnik
```

$$srednjaOcjena = \frac{\sum_{i=1}^{brojOcjena} ocjena_i}{brojOcjena}, \tag{6.10}$$

gdje varijabla "*srednjaOcjena*" sadrži prosjek svih ocjena, "*ocjena*" sadrži trenutno unesenu (i-tu) ocjenu, "*brojOcjena*" sadrži broj ocjena, odnosno broj do sada položenih ispita, a varijabla "i" predstavlja varijablu za odabir pojedine ocjene za izračun zbroja, odnosno kontrolnu varijablu. Na osnovi izraza 6.10 moguće je napraviti popis potrebnih varijabli koji je prikazan u tablici 6.6. Potrebno je primijetiti kako je dodana još jedna varijabla za izračun zbroja svih ocjena. Prosjek je općenito broj s plivajućim zarezom pa je za tip kod varijable "*srednjaOcjena*" potrebno odabrati broj s plivajućim zarezom, dok ostale varijable mogu biti cijeli brojevi.

Skica rješenja ovog problema dana je pseudokôdom 6.24. U prvom dijelu pseudokôda radi se deklaracija i inicijalizacija varijabli. Zatim se unosi broj položenih ispita kako bi se mogla inicijalizirati petlja "*for*". Nakon unosa podataka počinje petlja i njeno tijelo se izvršava za svaki položeni ispit, odnosno za svaku unijetu ocjenu. To znači da kontrolna varijabla "i" dobiva početnu vrijednost 1 i povećava se za 1 dok ne dostigne vrijednost unijetu za broj ocjena. U tijelu

Ime varijable	Tip varijable	Značenje varijable
ocjena	cijeli broj	Ocjena iz pojedinog predmeta
srednjaOcjena	broj s plivajućim zarezom	Prosječna ocjena
brojOcjena	cijeli broj	Količina do sada položenih ispita
zbrojOcjena	cijeli broj	Zbroj svih ocjena
i	cijeli broj	Kontrolna varijabla petlje "*for*"

Tablica 6.6: Popis varijabli idejnog programskog rješenja za izračun prosječne ocjene.

Pseudokôd 6.24: Izračun prosječne ocjene.

Ulaz:
Izlaz:
Deklaracija:
    ocjena, srednjaOcjena, brojOcjena, zbrojOcjena, i
Inicijalizacija:
    zbrojOcjena := 0
Ispis:
    "Unesite broj položenih ispita >"
Unos:
    brojOcjena
za $i := 1$ **do** *brojOcjena* **činiti**
  Ispis:
    "Unesite ocjenu >"
  Unos:
    ocjena
  zbrojOcjena := zbrojOcjena + ocjena
kraj za
Izračunaj:
    srednjaOcjena := zbrojOcjena / brojOcjena
Ispis:
    "Srednja ocjena do sada položenih ispita iznosi " + srednjaOcjena

petlje se radi unos pripadne ocjene i njena vrijednost se dodaje dosadašnjem zbroju ocjena. Nakon što su unijete sve ocjene, petlja "*for*" završava i sada je moguće odrediti srednju ocjenu dijeljenjem izračunatog zbroja svih ocjena njihovim brojem. Na kraju se radi ispis izračunatog prosjeka ocjena u pripadnoj poruci operateru.

6.7 Zadaci za samostalan rad

Sljedeći zadaci predstavljaju neke od osnovnih problema za usvajanje programiranja. Prilagođeni su gradivu kojeg studenti uče u sklopu nastave na Fakultetu prometnih znanosti. Osim za vježbanje izrade ideje programa, odnosno pseudokôda, ovi zadaci se mogu koristiti i za usvajanje izrade dijagrama toka te pisanja programa u programskom jeziku C♯. Programi za čije rješenje je potrebno iskoristiti petlju moguće je riješiti u tri inačice, odnosno korištenjem sve tri petlje. Studentima se preporuča izrada i testiranje svake inačice rješenja.

■ Zadatak 6.1

Izradite program za izračun Coulomb-ove sile između dva naboja smještenih u nekom mediju. U ispisu rezultata potrebno je također naznačiti radi li se o privlačnoj ili odbojnoj sili.

■ Zadatak 6.2

Potrebno je izraditi program za izračun nadomjesnog strujnog izvora "S" paralelno spojenih strujnih izvora. Pretpostavite da su u paralelnom spoju strujni izori različitog polariteta (smjera struje) i da svaki izvor ima svoj pripadni unutrašnji otpor. Parametre nadomjesnog strujnog izvora je potrebno ispisati u sklopu pripadne poruke operateru.

■ Zadatak 6.3

Otpremnik vlakova na željezničkom kolodvoru ispisuje na oglasnoj ploči vrijeme putovanja vlaka između dva odredišta. Za pomoć mu je potrebno izraditi program koji će na osnovu unesenih vremena odlaska i dolaska izračunati vrijeme putovanja. Pretpostavite da se vrijeme dolaska i odlaska unosi u obliku "sati:minute" te da se sva putovanja odvijaju unutar istog dana.

■ Zadatak 6.4

Suradnik na parkingu dobio je zadatak poredati tri vozila po njihovoj duljini. U tu svrhu mu je potrebno napisati program koji će pronaći najdulje vozilo između tri vozila. Za rješenje ovog problema potrebno je iskoristiti grananje. Rezultat je potrebno ispisati u sklopu odgovarajuće poruke operateru.

■ Zadatak 6.5

Djelatnik na pošti sortira pakete po njihovoj masi. Kako bi mu se olakšao posao, potrebno mu je izraditi program koji će pronaći najmanju masu između "P" paketa koje je suradnik dobio za sortiranje. Rezultat je potrebno ispisati u sklopu odgovarajuće poruke operateru.

■ Zadatak 6.6

Potrebno je izraditi program koji će izračunati umnožak brojeva koji su unutar zadanog intervala. Pretpostavite da operater zna količinu brojeva za obradu te donju i gornju granicu intervala brojeva. Obje granice su uključene u interval. Rezultat je potrebno ispisati u sklopu odgovarajuće poruke operateru.

■ Zadatak 6.7

Potrebno je izraditi program koji će izračunati ukupnu masu svih paketa čija se masa nalazi izvan zadanog intervala. Pretpostavite da operater zna količinu paketa za obradu te donju i gornju granicu intervala mase paketa. Donja granica mase nije uključena dok je gornja granica mase paketa uključena u interval. Rezultat je potrebno ispisati u sklopu odgovarajuće poruke operateru.

■ Zadatak 6.8

Avio-kompanija radi analizu starosti svoje flote aviona. U tu svrhu je potrebno izraditi program koji će za cijelu flotu aviona odrediti najmanju, najveću i prosječnu starost aviona.

■ Zadatak 6.9

Prometni policajac radi analizu poštivanja ograničenja brzine na određenom odsjeku cestovne prometnice. Tijekom dana je izmjerena brzina za ukupno "V" vozila. Za svaku izmjerenu brzinu potrebno je u pripadnoj poruci ispisati dali je vozilo poštivalo ograničenje, nalazi li se u dopuštenom području prekoračenja brzine ili je brzina veća od dopuštenog odstupanja. Pretpostavite da se dopušta odstupanje od 10 $[km/h]$.

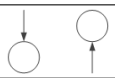
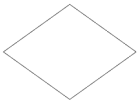
Poglavlje 7

Dijagrami toka

U prethodnom poglavlju (6) predstavljen je pseudokôd kao način kreiranja idejne skice programa, odnosno opisa pojedinog algoritma. Kako za izradu pseudokôda nema normiranih smjernica i kako se pseudokôd zasniva na ključnim riječima ponekad vezanim za pojedini jezik, pseudokôd nije uvijek najbolji način za prikaz idejne skice za izradu programa u multinacionalnim okruženjima. Dio problema se može riješiti tako da se u objavi opisa algoritma koriste ključne riječi iz engleskog jezika, no u slučaju jednostavnijih programa je najbolje idejnu skicu prikazati grafički. Za grafički prikaz idejne skice programa definirani su dijagrami toka. U njima se izvršavanje pojedine naredbe prikazuje pripadnim grafičkim simbolom tako da su dijagrami toka razumljiviji od pseudokôda u smislu prikaza postupaka i slijeda izvođenja naredbi. Također se dijagrami toka mogu koristiti za opis različitih procedura kao što su npr. upis studija, objašnjenje ispunjavanja porezne prijave, prikaz puštanja u pogon nekog tehničkog uređaja i sl. u drugim područjima. Na taj način dijagrami toka predstavljaju univerzalniji i jednostavniji opis od pseudokôda.

7.1 Simboli dijagrama toka

U dijagramu toka se naredbe programa ili cijele procedure predstavljaju grafičkim simbolima odnosno blokovima. Radi lakšeg dijeljenja gotovih dijagrama toka između suradnika na projektu izgled blokova je propisan normom [14], a dopunjavan je nakon izdavanja norme blokovima za izradu idejne skice složenijih programa. Tako danas postoje dijagrami toka za opis aktivnosti, poslovnih procesa, protoka podataka, upravljački dijagrami i sl. U sklopu ove skripte objasniti

Simbol bloka	Ime bloka	Opis bloka
	Terminator	Blok za oznaku početka i završetka dijagrama toka
	Strelica smjera	Označava smjer izvršavanja dijagrama toka
	Poveznica	Blokovi za povezivanja dijelova dijagrama toka ukoliko ne stane na jednu stranicu
	Unos	Blok za unos, odnosno pridruživanje vrijednosti varijabli u program
	Ispis	Blok za ispis vrijednosti varijable iz programa na zaslon računala
	Grananje	Blok za usporedbu vrijednosti te grananje izvođenja dijagrama toka ovisno o ispunjenju uvjeta
	Obrada	Blok za izvršavanje naredbi obrade podataka odnosno pridruživanja vrijednosti varijabli
	Inicijalizacija	Blok za postavljanje početnih vrijednosti varijabli prije složenijih operacija
	Proces	Blok za pozivanje složenijeg programskog bloka ili potprograma

Tablica 7.1: Pregled blokova za izradu dijagrama toka.

će se samo dijagrami toka vezani za izradu računalnih programa. U tablici 7.1 dan je popis blokova s pripadnim grafičkim simbolom koji će biti korišteni za izradu dijagrama toka.

Svaki dijagram toka počinje i završava blokom terminator u koji se stavlja ključna riječ "START" ili "POČETAK" kao oznaka početka dijagrama toka te ključna riječ "END" ili "KRAJ" kao oznaka završetka dijagrama toka. Blok terminator ima samo jedan izlaz u slučaju oznake početka, odnosno samo jedan ulaz u slučaju oznake završetka dijagrama toka. Svaki dijagram toka može imati samo jedan početak i samo jedan završetak. Blokovi se međusobno povezuju strelicama tako da je uvijek označen smjer izvršavanja dijagrama toka koji odgovara slijedu izvođenja naredbi programa. Smjer strelice je uvijek od bloka terminator koji označava početak dijagrama toka prema bloku terminator koji označava završetak dijagrama toka.

U slučaju da dijagram toka postane prevelik i ne stane više na jednu stranicu koristi se blok poveznica. Koristi se na način da se u prvi blok poveznice, koji ima jedan ulaz, unese numerička vrijednost. Unesena numerička vrijednost predstavlja jedinstvenu oznaku. Ona se unese i u blok poveznice koji ima jedan

izlaz, a taj blok onda predstavlja logički nastavak dijagrama toka. Tako se tijekom analize dijagrama toka blokovi poveznice s istim numeričkim oznakama smatraju povezanima, odnosno označavaju isto mjesto u dijagramu toka.

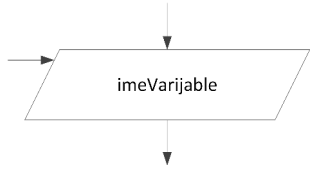
Blok za inicijalizaciju i blok za pozivanje potprograma koriste se u složenijim dijagramima toka kako bi se olakšala njihova analiza te povećala preglednost. U blok za inicijalizaciju unose se naredbe pridruživanja početnih vrijednosti varijabli koje su bitne za naredni programski odsječak. Pozivanje potprograma se radi na način da se u blok proces unese ime pozvanog potprograma, a dijagram toka pozvanog potprograma se da u prilogu. Ostali blokovi su objašnjeni detaljnije u nastavku.

7.2 Unos i ispis podataka

Blokovi za unos i ispis podataka služe programu za komunikaciju sa svojom okolinom. Komunikacija s okolinom uključuje unos vrijednosti za obradu u programu te ispis rezultata izvršavanja programa. Okolina programa može biti operater, drugi program ili neka udaljena vanjska jedinica kao što su to npr. različita osjetila za mjerenje količine prometa. U sklopu ove skripte obraditi će se samo komunikacija s operaterom.

7.2.1 Unos podataka

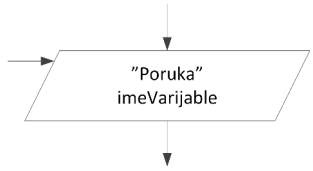
Blok za unos podataka služi za unos vrijednosti pojedine varijable. Najjednostavniji način korištenja ovog bloka je da se u blok unese ime varijable čija se vrijednost želi unijeti kao što je to prikazano na slici 7.1. Moguće je unijeti i vrijednost za više varijabli. Tada se u blok unesu imena svih varijabli kojima se želi unijeti vrijednost. Prilikom unosa prva se unesena vrijednost pridružuje prvoj varijabli, druga unesena vrijednost drugoj varijabli i tako redom dok se ne unesu vrijednosti svih varijabli.



Slika 7.1: Unos vrijednosti jedne varijable.

Prilikom unosa podataka za obradu, preporučljivo je operateru ispisati i pripadnu poruku kako bi mu se olakšalo korištenje programa, odnosno obavijestiti

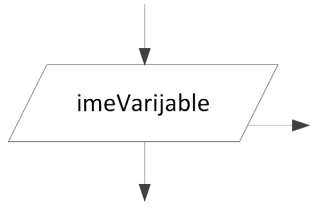
korisnika programa o tome koji se podatak trenutno unosi. To je također moguće korištenjem bloka za unos pri čemu se tekst poruke operateru označava dvostrukim navodnicima. Na slici 7.2 prikazan je primjer unosa vrijednosti varijable uz ispis poruke. Prilikom izvođena dijagrama toka prvo se ispiše poruka navedena unutar dvostrukih navodnika, a na kraju poruke na zaslonu se prikaže kursor. Izvođenje dijagrama toka se nastavlja nakon što operater unese traženu vrijednost. Vrijednost koju operater unese se sprema u varijablu navedenu u bloku za unos podataka. Na ovaj način moguće je unijeti vrijednost samo jedne varijable po bloku za unos podataka.



Slika 7.2: Unos vrijednosti jedne varijable uz ispis poruke operateru.

7.2.2 Ispis podataka

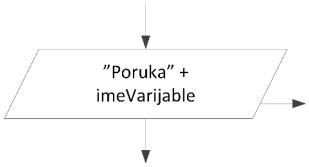
Blok za ispis podataka omogućuje ispis vrijednosti dobivenih izvršavanjem programa. Pri tome je moguće ispisati samo vrijednost pojedine varijable ili kombinaciju poruke te vrijednosti jedne ili više varijabli. Poruka koja se želi ispisati navodi se unutar dvostrukih navodnika, a za spajanje poruke i vrijednosti varijable koristi se operator "+". On u ovom slučaju spaja više znakovnih nizova u jedan pri čemu se vrijednost varijable za potrebe ispisa automatski pretvara u niz znakova. Na slici 7.3 prikazan je primjer ispisa vrijednosti jedne varijable. U slučaju da postoji zahtjev za ispis vrijednosti više varijabli potrebno ih je navesti unutar bloka za ispis. Pri tome se njihove vrijednosti na zaslonu računala ispisuju redoslijedom kako su navedene unutar bloka za ispis.



Slika 7.3: Ispis vrijednosti jedne varijable.

U slučaju ispisa vrijednosti varijable uz pripadnu poruku potrebno je unutar

bloka navesti i poruku te ime varijable za ispis. Moguće je ispisati vrijednosti više varijabli pri čemu se one dodaju operatorom "+" kako bi se spojile s ostatkom poruke za ispis. Na slici 7.4 prikazan je primjer ispisa vrijednosti jedne varijable uz pripadnu poruku operateru.



Slika 7.4: Ispis vrijednosti jedne varijable uz ispis poruke operateru.

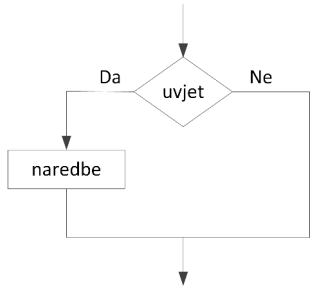
7.3 Grananja

Kao što je objašnjeno u potpoglavlju 6.4, grananje omogućuje donošenje odluke u programu, odnosno odabir između dvije ili više mogućnosti za nastavak izvršavanja programa. U dijagramu toka se za definiranje grananja koristi blok koji ima jedan ulaz i dva izlaza, pri čemu je jedan izlaz označen s "DA", a drugi izlaz označen je s "NE". Unutar bloka se navodi uvjet grananja koji kao rezultat daje logičku vrijednost. Ovisno o rezultatu uvjeta, odnosno ovisno o tome je li uvjet ispunjen ili ne se izvodi grana "DA" (uvjet ispunjen) ili grana "NE" (uvjet nije ispunjen) bloka grananja. Korištenjem jednog ili više blokova grananja moguće je izvesti sva grananja objašnjena u potpoglavlju 6.4.

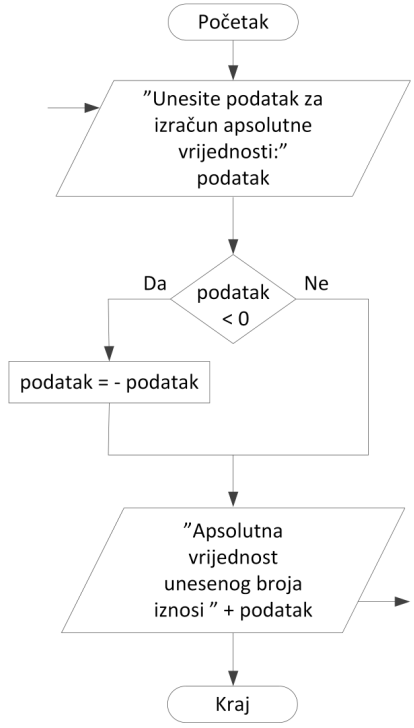
7.3.1 Grananje "if"

Grananje "*if*" predstavlja najjednostavnije grananje koja ima naredbe samo uz "DA" izlaz bloka grananja. Na slici 7.5 prikazan je princip izrade dijagrama toka tog najjednostavnijeg grananja. Potrebno je primijetiti da u slučaju ovog grananja grana izlaz "NE" ne sadrži naredbe.

Kao primjer dijagrama toka za primjenu ovog najjednostavnijeg grananja može poslužiti izračun apsolutne vrijednosti broja dan pseudokôdom 6.5. Dijagram toka ovog pseudokôda prikazan je slikom 7.6. Objašnjenje izvršavanja ovog dijagrama toka analogno je objašnjenju izvršavanja pseudokôda 6.5 pri čemu se izvršavanje može pratiti gledajući strelice koje definiraju protok podataka, odnosno smjer izvršavanja dijagrama toka.



Slika 7.5: Dijagram toka za grananje "*if*".

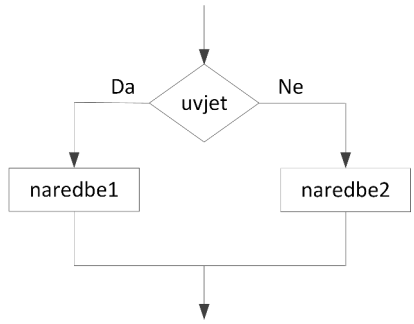


Slika 7.6: Dijagram toka za izračun apsolutne vrijednosti broja.

7.3.2 Grananje "if - else"

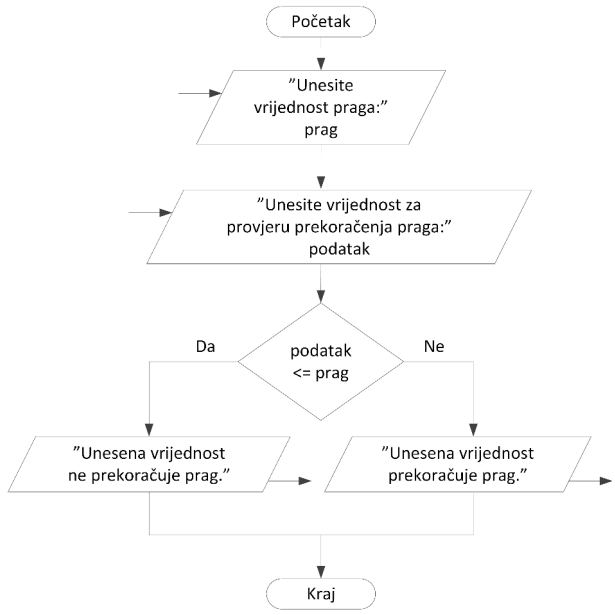
Grananje "*if - else*" predstavlja složenije grananje pri kojem je iskorištena i "NE" grana izlaza bloka za grananje. Kod ovog grananja su definirane naredbe i za slučaj da uvjet nije ispunjen. Na slici 7.7 prikazan je princip izrade dijagrama toka ovog grananja. Potrebno je primijetiti da u slučaju ovog grananja grana

uz izlaz "NE" sada sadrži naredbe za razliku od grananja "*if*". U pojedinim granama može biti samo jedna ili više naredbi.



Slika 7.7: Dijagram toka za grananje "*if - else*".

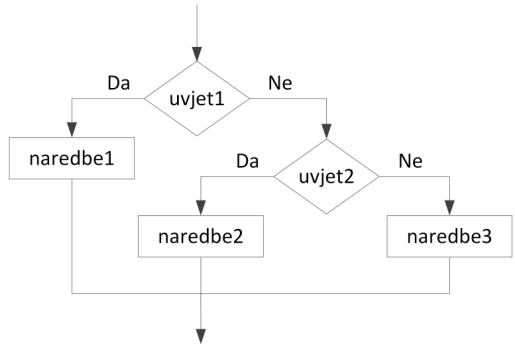
Kao primjer dijagrama toka za primjenu ovog grananja može poslužiti primjer provjere prekoračenja praga dan pseudokôdom 6.7. Dijagram toka ovog pseudokôda prikazan je slikom 7.8. Objašnjenje izvršavanja ovog dijagrama toka analogno je objašnjenju izvršavanja pseudokôda 6.7. Koriste se ista imena varijabli tako da je uz objašnjenje potrebno samo pratiti smjer izvršavanja dijagrama toka.



Slika 7.8: Dijagram toka za provjeru prekoračenja praga.

7.3.3 Grananje "if - else if - else"

Najsloženije grananje "*if - else if - else*" omogućava provjeru više uvjeta. Za svaki uvjet potrebno je iskoristiti zaseban blok za grananje, a svaki sljedeći blok za grananje uvijek dolazi u "NE" granu prethodnog bloka grananja. Time se osigurava provjeravanje dodatnih uvjeta samo ako prethodni uvjet nije bio ispunjen. Na slici 7.9 prikazan je princip izrade dijagrama toka za ovo grananje. Potrebno je primijetiti kako se na kraju cijele strukture grananja sve grane izvršavanja ponovno spajaju u jednu granu. To znači da se izvrši samo jedna grana od svih mogućih u grananju i nakon toga se izvršava prvi blok koji dolazi nakon cijele strukture grananja. Ovo spajanje grana može poslužiti i za brzu provjeru ispravnosti dijagrama toka koji koristi grananje kako dijagram toka uvijek završava blokom "KRAJ" ili "END" i može postojati samo jedan blok završetka dijagrama toka.

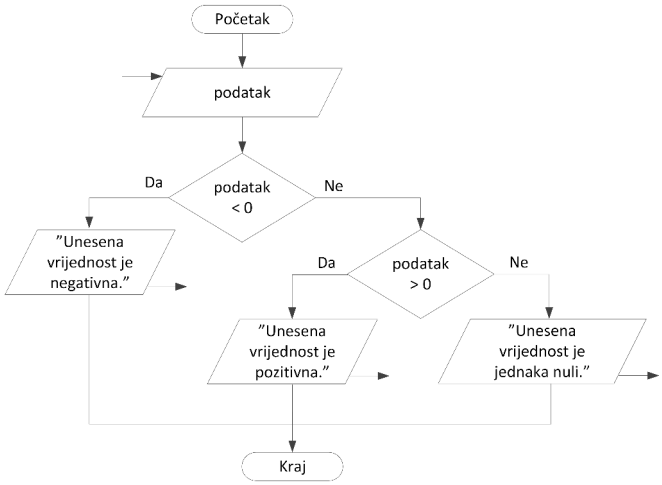


Slika 7.9: Dijagram toka za grananje "*if - else if - else*".

Kao primjer dijagrama toka za primjenu ovog grananja može poslužiti primjer provjere predznaka broja dan pseudokôdom 6.9. Dijagram toka ovog pseudokôda prikazan je slikom 7.10. Objašnjenje izvršavanja je analogno objašnjenju pseudokôda 6.9. Bitno je primijetiti da dijagram toka sadrži dva bloka grananja kako bi se provjerio predznak, odnosno je li broj veći od nule ili manji od nule, a grana kada niti jedan uvjet nije ispunjen označava slučaj kada je podatak jednak nuli. Pri tome je prvi blok grananja vezan za provjeru osnovnog uvjeta, a drugi blok za provjeru dodatnog uvjeta.

7.4 Petlje

Za kreiranje petlji ne postoje posebni blokovi u dijagramu toka, već se koristi blok grananja za provjeru uvjeta petlje i povrat nakon izvršavanja blokova koji



Slika 7.10: Dijagram toka za provjeru predznaka broja.

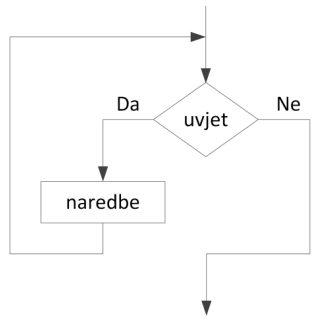
čine tijelo petlje na ponovnu provjeru uvjeta petlje. Pri tome je uvjet petlje općenito predstavljen logičkim izrazom koji u sebi može sadržavati i aritmetički dio. Blokovi tijela petlje dolaze u "DA" granu bloka grananja, a "NE" grana predstavlja završetak petlje.

7.4.1 Petlja "while"

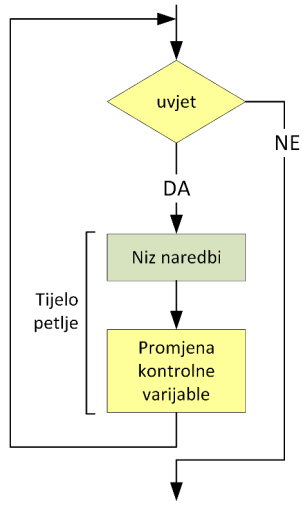
Kako je već objašnjeno u potpoglavlju 6.5, petlja "*while*" provjerava uvjet prije izvršavanja tijela petlje i tijelo petlje se izvršava dok god je uvjet ispunjen. Kada uvjet petlje više nije ispunjen izvršava se prva naredba koja dolazi nakon tijela petlje. Struktura dijagrama toka za kreiranje petlje "*while*" prikazana je na slici 7.11. Za provjeru uvjeta petlje "*while*" koristi se blok grananja. U njegovu "DA" granu dolaze blokovi tijela petlje. U tijelu petlje može biti jedan ili više blokova.

Prilikom izrade tijela petlje "*while*" bitno je osigurati da se u tijelu petlje promijeni kontrolna varijabla koja se provjerava u uvjetu na početku petlje. Radi lakšeg raspoznavanja te naredbe je na slici 7.12 tijelo petlje "*while*" podijeljeno na dva dijela. Prvi dio sadrži naredbe odnosno blokove vezane za obradu podataka u tijelu petlje, a zadnja naredba odnosno blok označen svjetlo žutom bojom sadrži naredbu za promjenu kontrolne varijable koja se provjerava u bloku grananja na početku petlje. Bez te naredbe odnosno bloka neće se nikad ispuniti uvjet završetka petlje i nastaje tzv. beskonačna petlja.

Kao primjer za implementaciju petlje "*while*" može poslužiti izračun zbroja

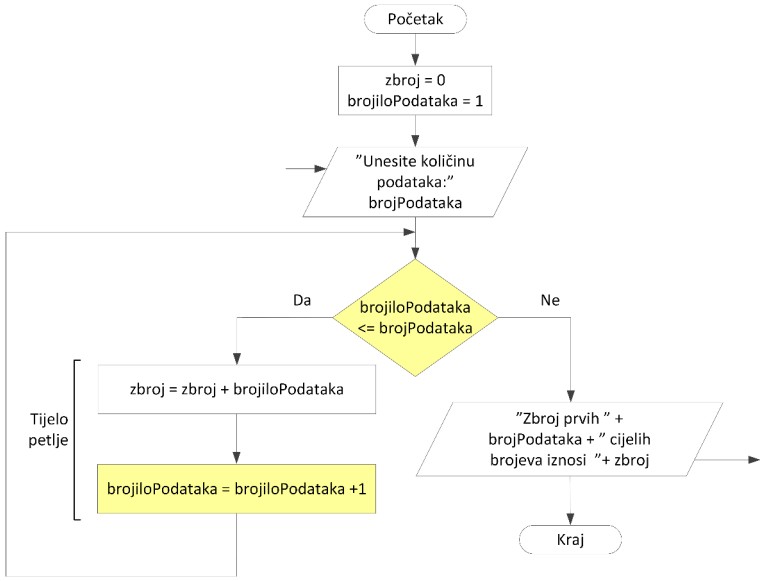


Slika 7.11: Dijagram toka za implementaciju petlje "*while*".



Slika 7.12: Tijelo petlje "*while*" s istaknutim dijelovima.

prvih "n" cijeli pozitivnih brojeva dan pseudokôdom 6.13. Pripadni dijagram toka prikazan je slikom slici 7.13. Objašnjenje izvršavanja ovog dijagrama toka analogno je objašnjenu izvršavanja pseudokôda 6.13. Moguće je primijetiti kako se tijelo petlje sastoji od dva bloka za obradu. Prvi blok uvećava varijablu "*zbroj*" tako da joj doda sljedeći cijeli broj u nizu. Drugi blok povećava vrijednost varijable "*brojiloPodataka*" za 1 i ta se naredba odnosi na promjenu vrijednosti kontrolne varijable. Moguće je primijetiti da se varijabla "*brojiloPodataka*" uspoređuje s vrijednošću varijable "*brojPodataka*" u bloku grananja na početku petlje. Iz tog razloga je taj blok označen svjetlo žutom bojom kao i blok za provjeru uvjeta. Bez te naredbe petlja ne bi nikada završila.



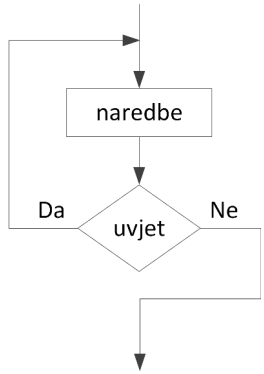
Slika 7.13: Dijagram toka za izračun zbroja prvih "n" cijelih brojeva.

7.4.2 Petlja "do - while"

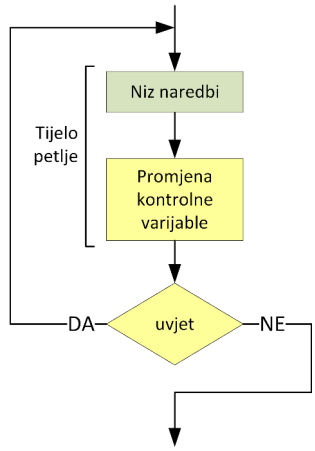
Kako je objašnjeno u potpoglavlju 6.5 petlja "*do - while*" provjerava uvjet nakon izvršavanja tijela petlje. Tijelo petlje se izvršava dok god je uvjet ispunjen pri čemu je sada osigurano da će se tijelo petlje izvršiti barem jednom. Kada uvjet petlje više nije ispunjen, izvršava se prva naredba koja dolazi nakon tijela petlje. Struktura dijagrama toka za kreiranje petlje "*do - while*" prikazana je na slici 7.14. Za provjeru uvjeta petlje "*do - while*" ponovno se koristi blok grananja koji se sada nalazi iza tijela petlje. U ovoj petlji "DA" grana služi za ponovni povratak na početak tijela petlje. U tijelu petlje može biti jedan ili više blokova.

Ponovno je potrebno u tijelo petlje staviti naredbu, odnosno blok, koji će promijeniti kontrolnu varijablu koja se provjerava u uvjetu. Radi lakšeg raspoznavanja te naredbe, odnosno bloka, je na slici 7.15 tijelo petlje razdvojeno ponovno na dva dijela. Prvi dio se odnosi na obradu podataka koju je potrebno napraviti u tijelu petlje. Drugi dio tijela petlje je blok koji mijenja iznos kontrolne varijable koja se provjerava u bloku grananja na kraju tijela petlje. Taj blok je označen svjetlo žuto kao i blok grananja. Kao i u slučaju petlje "*while*" taj dio je potreban kako bi se osiguralo da petlja može završiti. Bez njega ponovno bi nastala tzv. beskonačna petlja koja nikada ne bi završila.

Kao primjer implementacije petlje "*do - while*" može poslužiti rješenje problema izračuna umnoška prvih "n" cijelih brojeva danog pseudokôdom 6.15.

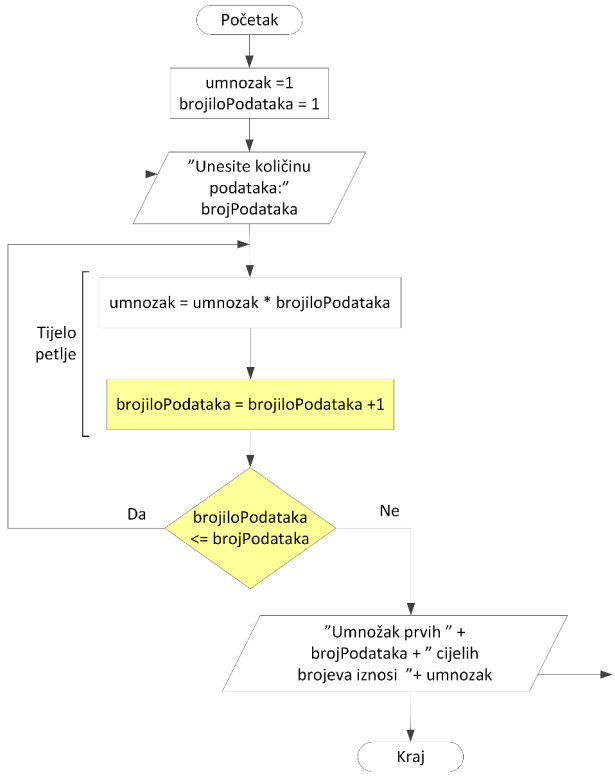


Slika 7.14: Dijagram toka za implementaciju petlje "*do - while*".



Slika 7.15: Tijelo petlje "*do - while*" s istaknutim dijelovima.

Pripadni dijagram toka prikazan je slikom 7.16. Objašnjenje izvršavanja ovog dijagrama toka analogno je objašnjenju izvršavanja pseudokôda 6.15. Na slici 7.16 moguće je primijetiti kako se uvjet petlje sada provjerava nakon tijela petlje osiguravajući da će se tijelo petlje izvršiti najmanje jednom. U sklopu tijela petlje se ponovno nalazi naredba odnosno blok za promjenu vrijednosti kontrolne varijable "*brojiloPodataka*". Blok je označen žuto kao i blok za provjeru uvjeta petlje, odnosno vrijednosti kontrolne varijable. Grana "DA" uzrokuje ponovno izvršavanje tijela petlje i u ovom slučaju vraća izvođenje dijagrama toka na početak tijela petlje. Grana "NE" označava kraj petlje i kao sljedeća naredba, odnosno blok, se izvodi ispis rezultata izračunatog umnoška.

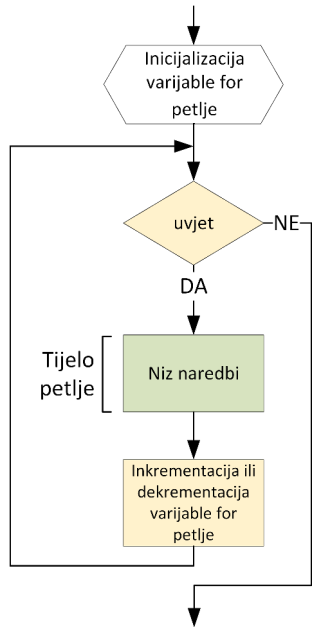


Slika 7.16: Dijagram toka za izračun umnoška prvih "n" cijelih brojeva.

7.4.3 Petlja "for"

Kako je objašnjeno u potpoglavlju 6.5 petlja "*for*" sadrži u svojoj definiciji inicijalizaciju kontrolne varijable, uvjet i naredbu za promjenu (inkrement ili dekrement) vrijednosti kontrolne varijable. Za implementaciju petlje "*for*" u sklopu dijagrama toka se koristi blok za inicijalizaciju kako bi se postavila početna vrijednost kontrolne varijable petlje "*for*", blok grananja za provjeru uvjeta i blok za obradu podataka u kojem se mijenja vrijednost kontrolne varijable. Između bloka grananja za provjeru uvjeta petlje "*for*" i bloka za obradu podataka radi promjene vrijednosti kontrolne varijable dolaze blokovi tijela petlje. Pripadni dijagram toka za implementaciju petlje "*for*" s istaknutim dijelovima prikazan je slikom 7.17. Vidljivo je da se blok inicijalizacije izvršava samo jednom prilikom pokretanja petlje "*for*".

Kao primjer implementacije petlje "*for*" može poslužiti izračun zbroja cijelih brojeva unutar zadanog intervala čije je rješenje dano pseudokôdom 6.18. Pripadni dijagram toka prikazan je slikom 7.18. Objašnjenje izvršavanja ovog

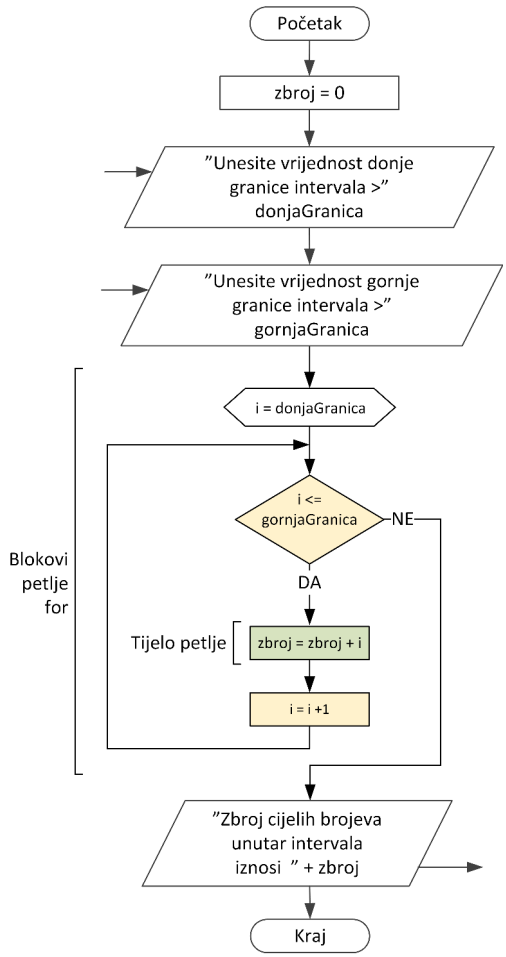


Slika 7.17: Dijagram toka za implementaciju petlje "*for*" s istaknutim dijelovima.

dijagrama toka odgovara objašnjenju izvršavanja pseudokôda 6.18. Na slici 7.18 moguće je primijetiti pojedine dijelove petlje "*for*" počevši blokom za inicijalizaciju u kojem se postavlja početna vrijednost kontrolne varijable "*i*", bloka za provjeru uvjeta, tijela petlje i bloka za inkrement kontrolne varijable petlje "*for*". Pri tome blokovi tijela petlje (označeni zelenom bojom) dolaze između bloka grananja i bloka obrade za inkrement kontrolne varijable. Tijelo petlje se sada sastoji samo od blokova za obradu podataka čime se pojednostavljuje izrada same petlje "*for*" iz postojećeg pseudokôda.

7.5 Aplikacija Raptor

Prilikom izrade dijagrama toka preporučljivo je izrađeni dijagram toka provjeriti. U tu svrhu je potrebno generirati manji testni skup podataka koji obuhvaća sve potencijalne karakteristične ili problematične slučajeve. Jednostavniji dijagrami toka mogu se provjeriti i ručno dok je za složenije dijagrame toka uputno koristiti programske alate. Jedan od takvih programskih alata je i aplikacija Raptor, besplatno dostupna za preuzimanje na [2]. Aplikacija Raptor omogućuje implementaciju dijagrama toka i njihovo testiranje [15]. Prilikom izvođenja dijagrama toka na lijevoj se strani korisničkog sučelja ispisuju trenutne vrijed-

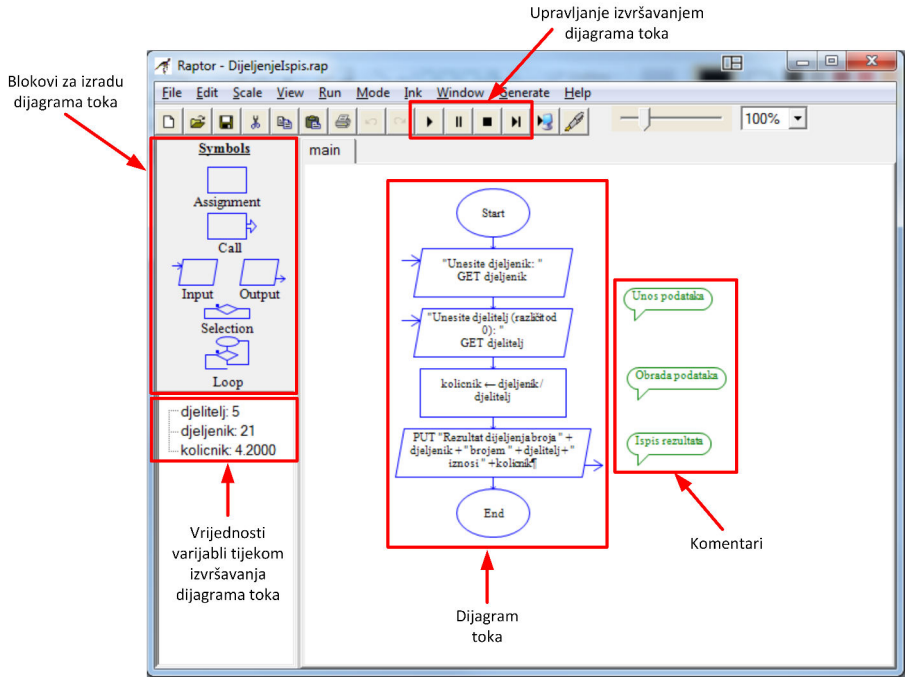


Slika 7.18: Dijagram toka za izračun zbroja cijelih brojeva unutar intervala.

nosti varijabli, što omogućuje jednostavnije ispravljanje implementiranog dijagrama toka. Za detaljniju analizu dijagrama toka moguće je njegovo izvršavanje blok po blok pri čemu korisnik pritiskom na pripadnu tipku pokreće izvršavanje sljedećeg bloka. Blok koji se trenutno izvršava označen je zelenom bojom.

Grafičko sučelje aplikacije Raptor prikazano je slikom 7.19. Sastoji od dijelova uobičajenih za Windows aplikacije i specifičnih dijelova za izradu dijagrama toka. Crvenim pravokutnicima označeni su bitni dijelovi grafičkog sučelja za izradu dijagrama toka kao što su: blokovi za izradu dijagrama toka, tipke za upravljanje izvršavanjem dijagrama toka, izrađeni dijagram toka, komentari pojedinih blokova dijagrama toka i trenutne vrijednosti varijabli dijagrama toka. Prikazane vrijednosti varijabli se mijenjaju kako se izvršavaju blokovi

dijagrama toka i odgovaraju trenutnim vrijednostima varijabli ovisno o izvođenju dijagrama toka. Prilikom prvog korištenja neke varijable ona se automatski deklarira tako da nije potrebno raditi posebnu deklaraciju varijabli. To olakšava testiranje koncepta programskog rješenja kako se tip varijable automatski prilagođava vrijednosti koja se želi spremiti u varijablu prilikom njenog prvog korištenja, odnosno deklaracije.

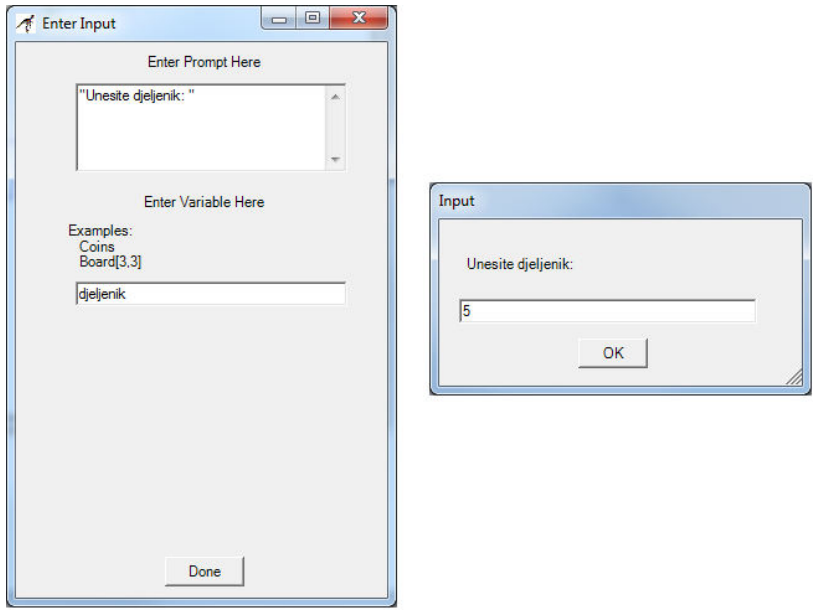


Slika 7.19: Grafičko sučelje aplikacije Raptor.

U lijevom dijelu slike 7.19 prikazan je prostor s blokovima za izradu dijagrama toka (engl. "Symbols"). Moguće je primijetiti da su na raspolaganju blok za obradu podataka (engl. "Assignment"), blok za pozivanje drugog dijagrama toka (engl. "Call"), blokovi za unos (engl. "Input") i ispis (engl. "Output") vrijednosti varijabli, blok za grananje (engl. "Selection") i blok za izradu petlji (engl. "Loop"). Blok za grananje sastoji se od bloka za provjeru iz kojeg izlaze dvije grane spojene u jednu radi nastavka dijagrama toka. Prilikom unosa strukture grananja blokovi se ubacuju u pripadnu granu poštujući oznake "Yes" (za "DA" granu, odnosno uvjet ispunjen) i "No" (za "NE" granu, odnosno uvjet nije ispunjen). Blok za izradu petlji sastoji se od bloka oznake početka petlje i bloka za grananje. Ovisno o inačici aplikacije Raptor grana koja se od bloka grananja vraća prema oznaci početka petlja može biti vezana za slučaj ispunjavanja uvjeta (engl. "Yes") ili neispunjavanja uvjeta (engl. "No"). Prema toj

konfiguraciji je potrebno i prilagoditi uvjet petlje pri čemu se za korištenje "NE" grane za izvršavanje tijela petlje uvjet petlje samo invertira (komplementira). To znači da usporedba "*manje od*" postaje "*veće ili jednako od*" i obrnuto. Za unos petlje "*while*" ili petlje "*for*", blokovi tijela petlje dolaze u granu koja se vraća od bloka grananja do oznake početka petlje. Blokovi tijela petlje "*do - while*" dolaze u granu između oznake početka petlje i bloka grananja.

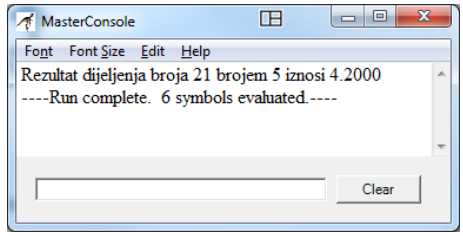
Na gornjem dijelu grafičkog sučelja aplikacije Raptor danog na slici 7.19 prikazane su i tipke za upravljanje izvršavanjem dijagrama toka. Prve tri tipke označavaju pokretanje izvršavanja, privremenog zaustavljanja izvršavanja do ponovnog pritiska iste tipka za nastavak izvršavanja i prijevremeno zaustavljanje izvršavanja dijagrama toka. Za učenje i analizu izvršavanja dijagrama toka posebno je korisna četvrta tipka koja omogućava izvršavanje bloka po blok. Pri tome se sljedeći blok po redu izvrši tek kada se pritisne navedena tipka. Nakon izvršavanja pripadnog bloka se u prostoru za prikaz vrijednosti varijabli prikažu osvježene vrijednosti varijabli. Time je omogućena detaljna analiza rezultata izvršavanja bloka, odnosno provjera ispravnosti dijagrama toka.



Slika 7.20: Grafičko sučelje za definiranje bloka za unos (lijevo) te unos vrijednosti varijable tijekom izvršavanja dijagrama toka (desno).

Kod korištenja bloka za unos vrijednosti varijabli potrebno ga je prvo prenijeti na željeno mjesto u dijagramu toka. To se radi korištenjem principa prenesi i ispusti (engl. "drag & drop"). Nakon toga je potrebno blok otvoriti radi unosa

poruke operateru te imena varijable. Pripadno grafičko sučelje za pripremu bloka za unos prikazano je na lijevoj strani slike 7.20. U gornje polje se unosi poruka kao niz znakova pri čemu dvostruki navodnici označavaju početak i kraj niza znakova, odnosno poruke za ispis operateru. U donje polje se unosi ime varijable u koju će se spremiti unesena vrijednost. Pomoću jednog bloka za unos moguće je unijeti vrijednost samo jedne varijable. Prilikom izvršavanja dijagrama toka operateru se prikaže grafičko sučelje za unos vrijednosti prikazano na desnoj strani slike 7.20. U gornjem dijelu tog grafičkog sučelja moguće je primijetiti poruku operateru, a vrijednost operater unosi u polje ispod poruke. Vrijednost se unosi potvrdom na tipku "OK". Nakon potvrde, nova vrijednost varijable biti će vidljiva u dijelu grafičkog sučelja aplikacije Raptor za prikaz trenutnih vrijednosti varijabli.



Slika 7.21: Grafičko sučelje glavne konzole za ispis rezultata izvođenja dijagrama toka.

Rezultat izvršavanja dijagrama toka se prikazuje u prozoru glavne konzole (engl. "MasterConsole") prikazane na slici 7.21. U njoj se ispisuju sve poruke i vrijednosti varijabli navedene u blokovima za ispis unutar dijagrama toka. Također se na ovoj konzoli ispisuje statistika izvršavanja dijagrama toka, odnosno broj blokova koji se izvršio. Pri tome se kod petlji pojedini blok može izvršiti više puta i svako izvršavanje bloka se broji.

7.6 Primjeri

U ovom potpoglavlju dani su primjeri rješavanja dijagrama toka. Pri tome su iskorišteni isti primjeri dani u potpoglavlju 6.6 za objašnjenje kreiranje pseudokôda. Za izradu dijagrama toka iskorištene su iste varijable tako da u nastavku neće biti ponovno navođen popis varijabli, već samo poveznica na pripadnu tablicu u potpoglavlju 6.6.

■ Primjer 7.1

Potrebno je izraditi dijagram toka programa koji će ispitati je li se uneseni cijeli broj nalazi u zadanom intervalu. Rezultat je potrebno ispisati u odgovarajućoj poruci operateru. Pri tome je donja granica intervala uključena, a gornja granica nije uključena u interval.

Rješenje:

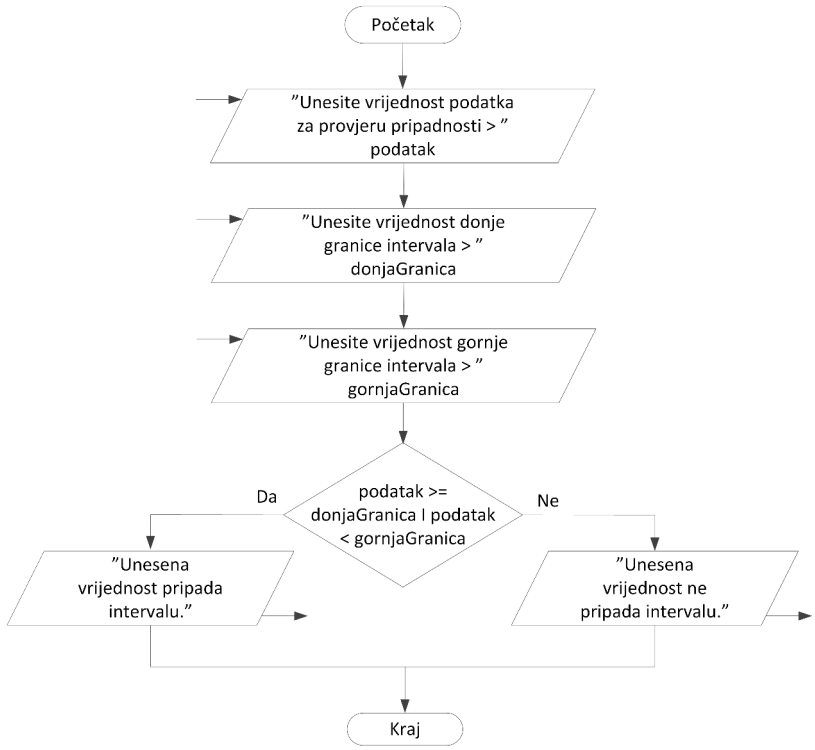
Varijable potrebne za rješavanje ovog problema navedene su u tablici 6.2, a koncept rješenja dan je pseudokôdom 6.19. Prevođenjem pseudokôda 6.19 u dijagram toka dobiva se dijagram toka prikazan na slici 7.22. Dobiveni dijagram toka se može podijeliti na dva dijela. Prvi dio se odnosi na unos potrebnih varijabli, a drugi dio se odnosi na provjeru uvjeta te ispis rezultata. Moguće je primijetiti kako je prilikom izrade dijagrama toka preskočen dio koji se odnosi na deklaraciju potrebnih varijabli definiran na početku pseudokôda 6.19. To je moguće kako se u dijagramu toka varijabla deklarira prilikom njenog prvog korištenja, odnosno pridruživanja vrijednosti.

Prilikom unosa vrijednosti varijabli se ujedno i operateru ispisuje poruka radi olakšanja korištenja dijagrama toka. U pseudokôdu 6.19 su u tu svrhu iskorištene dvije naredbe. Prvo naredba ispisa ispisuje poruku operateru, a nakon nje naredba za unos dohvaća vrijednost koju je operater unio. U dijagramu toka na slici 7.22 upotrijebljen je samo blok za unos u koji je unesena poruka koja će se ispisati operateru. Prilikom izvršavanja tog bloka prvo se ispiše poruka operateru, a nakon toga dijagram toka čeka da operater unese vrijednost.

Kod izrade uvjeta za blok grananja na slici 7.22 nisu navedene zagrade za odvajanje operacije usporedbe od logičke operacije I kako je to u pseudokôdu 6.19. Oba uvjeta su ispravna jer se prema prioritetu prvo izvrše operacije usporedbe, a nakon njih logička operacija. Ujedno operacije usporedbe kao svoj rezultat daju logičke vrijednosti, što je ispravan tip ulaznih podataka za logičku operaciju. Sam uvjet je izrađen na način da odgovor "DA" znači kako je podatak u zadanom intervalu, a odgovor "NE" kako podatak nije u zadanom intervalu. Tako je ispis rezultata stavljen u pripadnu granu. Nakon blokova za ispis rezultata, obje grane bloka za grananja spajaju se u jednu i dijagram toka završava.

■ **Primjer 7.2**

Mnogokut (poligon) je dio ravnine omeđen zatvorenom izlomljenom linijom ili manje formalno dio ravnine omeđen ravnim dužinama koje možemo nacrtati, a da ne podižemo olovku. Opseg mnogokuta jednak je zbroju duljina svih njegovih stranica. Potrebno je napraviti program u obliku dijagrama toka koji će izračunati opseg mnogokuta korištenjem strukture petlje "while".



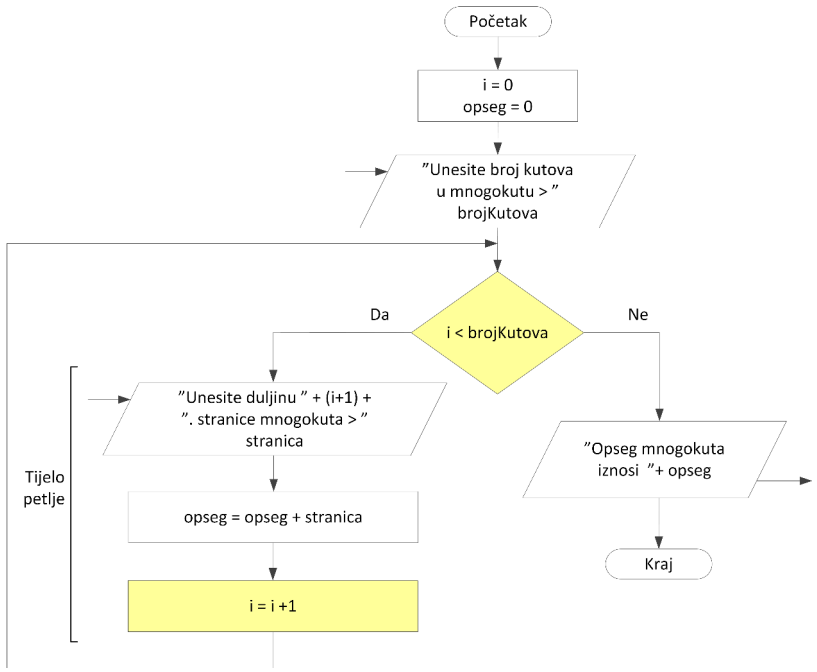
Slika 7.22: Dijagram toka za provjeru pripadnosti intervalu.

Rješenje:

Varijable potrebne za rješavanje ovog problema navedene su u tablici 6.4, a koncept rješenja dan je pseudokôdom 6.22. Prevođenjem pseudokôda 6.22 u dijagram toka dobiva se dijagram toka prikazan na slici 7.23. Ovaj dijagram toka može se podijeliti na dio za inicijalizaciju varijabli, unos varijabli, petlju "*while*" i ispis rezultata.

Tijelo petlje "*while*" počinje blokom grananja i u njegovu "DA" granu dolaze blokovi tijela petlje. Žutom bojom označeni su blokovi petlje "*while*" koji se odnose na provjeru uvjeta petlje i promjenu kontrolne varijable petlje. U slučaju ovog dijagrama toka radi se o kontrolnoj varijabli "*i*" koja označava broj trenutno unesenih duljina stranica mnogokuta. Nju je potrebno povećati nakon svakog unosa stranice. Nakon što se tijelo petlje izvrši, dijagram toka se ponovno vraća na točku prije provjera uvjeta petlje kako bi se tijelo petlje moglo ponovno izvršiti ako je uvjet ispunjen.

Samo tijelo petlje sadrži i izračun opsega mnogokuta. Opseg mnogokuta



Slika 7.23: Dijagram toka za izračun opsega mnogokuta.

se računa tako da se u tijelu petlje unesena duljina stranice doda postojećem zbroju duljina unesenih stranica. Tako varijabla "*opseg*" sadrži vrijednost opsega mnogokuta kad petlja "*while*" završi s radom. Nakon petlje je potrebno još samo napraviti ispis rezultata.

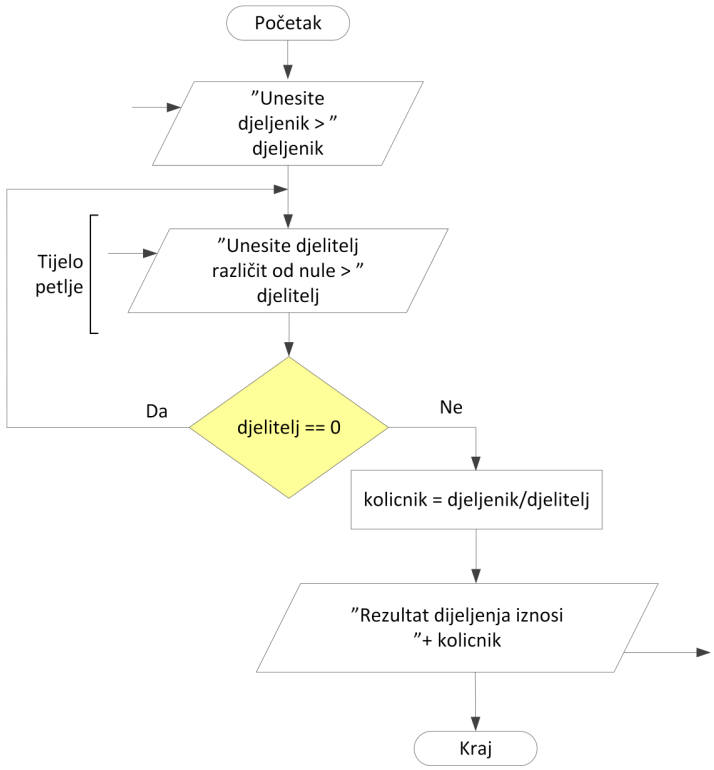
■ Primjer 7.3

Operacija dijeljenja jedna je od aritmetičkih operacija koja nije definirana za sve moguće vrijednosti djelitelja. Naime, dijeljenje s nulom nije definiramo što znači kako je operaciju dijeljenja moguće izvršiti samo u slučaju ako je djelitelj različit od nule. Potrebno je izraditi program koji će imati ugrađenu zaštitu kod unosa podataka zasnovanu na petlji "do - while".

Rješenje:

Varijable potrebne za rješavanje ovog problema navedene su u tablici 6.5, a koncept rješenja dan je pseudokôdom 6.23. Prevođenjem pseudokôda 6.23 u dijagram toka dobiva se dijagram toka prikazan na slici 7.24. Ovaj dijagram toka može se podijeliti na unos varijabli, petlju "*do - while*", izračun i ispis

rezultata.



Slika 7.24: Dijagram toka za dijeljenje dva broja s ugrađenom zaštitom.

Petlja "*do - while*" u ovom je slučaju iskorištena za provjeru ispravnosti unosa varijabli, odnosno provjeru ispravnog unosa varijable "*djelitelj*". Za provjeru ispravnosti unosa prvo je potrebno unijeti vrijednosti varijable, zatim ispitati ispravnost unesene vrijednosti te po potrebi ponoviti unos. U ovaj redoslijed operacija se petlja "*do - while*" izvrsno uklapa jer se kod nje prvo izvrši tijelo petlje, a nakon tijela petlje se ispita uvjet petlje. Kako je vidljivo na slici 7.24, tijelo petlje se sada sastoji samo od bloka za unos varijable "*djelitelj*". Ta varijabla je ujedno i kontrolna varijabla petlje. Tako je osigurano da će petlja u jednom trenutku završiti. Nakon što petlja završi s radom izračunava se rezultat dijeljenja i ispisuje se rezultat dijeljenja.

■ Primjer 7.4

Student Fakulteta prometnih znanosti želi se natjecati za stručnu praksu u inozemstvu. Za ispunjavanje prijavnice potreban mu je prosjek ocjena iz svih

do sada položenih ispita. Kako bi se studentu olakšao izračun prosječne ocjene potrebno je napraviti program u obliku dijagrama toka koji će srednju ocjenu izračunati korištenjem petlje "for".

Rješenje:

Varijable potrebne za rješavanje ovog problema navedene su u tablici 6.6, a koncept rješenja dan je pseudokôdom 6.24. Prevođenjem pseudokôda 6.24 u dijagram toka dobiva se dijagram toka prikazan na slici 7.25. Ovaj dijagram toka može se podijeliti na inicijalizaciju varijabli, unos podataka, petlju "*for*", izračun i ispis rezultata.

Na slici 7.25 označeni su blokovi koji spadaju u strukturu petlje "*for*". Vidljivo je kako ona počinje blokom za inicijalizaciju kontrole varijable petlje, zatim dolazi blok za provjeru uvjeta petlje (označen svjetlo žuto), tijelo petlje i blok za promjenu vrijednosti (inkrement) kontrolne varijable petlje. U slučaju petlje "*for*" promjena vrijednosti kontrolne varijable nije dio tijela petlje. U ovom slučaju se tijelo petlje sastoji od unosa nove ocjene te bloka obrade koji unesenu ocjenu pridoda postojećem zbroju do sada unesenih ocjena. Tako varijabla "*zbrojOcjena*" sadrži zbroj svih ocjena do sada položenih ispita nakon što petlja završi s radom. Nakon završetka petlje potrebno je samo izračunati srednju ocjenu (omjer zbroja svih ocjena i broja ocjena) te ispisati dobivenu vrijednost.

7.7 Zadaci za samostalan rad

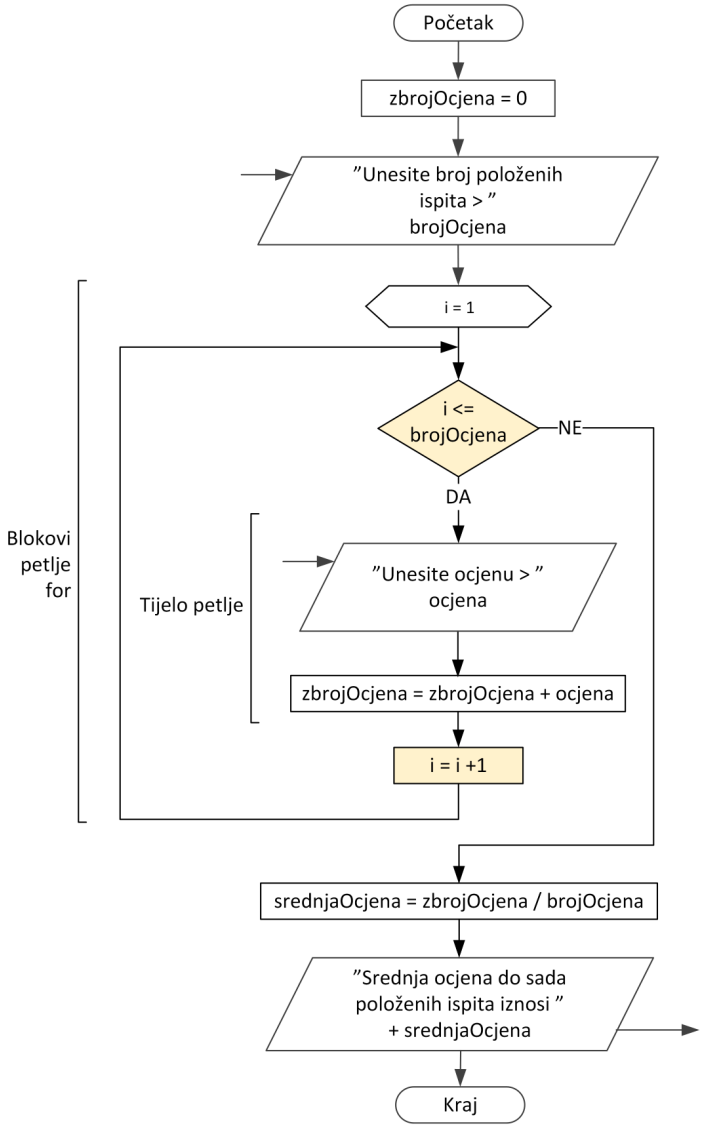
■ Zadatak 7.1

Vozilo ima spremnik goriva zapremnine Z litara i na prijeđenih 100 *kilometara vozilo potroši X litara goriva. Nacrtajte dijagram toka koji će ispisati doseg vozila uz unesene vrijednosti količine goriva u spremniku K i prosječne potrošnje goriva P.*

■ Zadatak 7.2

Potrebno je izraditi dijagram toka za dijeljenje dva broja. Napravite provjeru ispravnosti djelitelja te ispišite pripadnu poruku operateru ako nije moguće izvršiti operaciju dijeljenja.

■ Zadatak 7.3



Slika 7.25: Dijagram toka za izračun prosječne ocjene pomoću petlje "*for*".

Potrebno je izraditi dijagram toka za dijeljenje dva broja. Napravite provjeru ispravnosti djelitelja te ako nije moguće izvršiti operaciju dijeljenja ponoviti unos djelitelja. Pri pokretanju svake procedure novog unosa djelitelja zbog ispravka pogreške potrebno je ispisati pripadnu poruku operateru.

■ **Zadatak 7.4**

Nacrtajte dijagram toka koji će učitati 50 *cijelih brojeva i ispisati najmanji cijeli broj.*

■ **Zadatak 7.5**

Nacrtajte dijagram toka koji će učitati N cijelih brojeva i ispisati broj učitanih cijelih brojeva većih od 0*.*

■ **Zadatak 7.6**

Nacrtajte dijagram toka koji će iz 24 *unesena podatka o izmjerenoj gustoći prometnog toka izračunati prosječnu vrijednost gustoće prometnog toka.*

■ **Zadatak 7.7**

*Taksi tvrtka ima u vlasništvu ukupno N vozila od kojih su jedna skupina limuzine, a druga skupina kombi vozila. Nacrtajte dijagram toka koji će na osnovi unesenih oznaka vozila (npr. "*0*" - limuzina, "*1*" - kombi vozilo) izbrojati te ispisati količinu pojedinih vozila kao i pripadnu poruku kojih vozila ima više (npr. "Ima više limuzina od kombi vozila." i obrnuto, odnosno "Ima jednak broj limuzina i kombi vozila.").*

■ **Zadatak 7.8**

Sastavite dijagram toka za izračunavanje najmanjeg parnog broja u nizu od N prirodnih brojeva.

■ **Zadatak 7.9**

Sastavite dijagram toka za izračunavanje srednje vrijednosti neparnih brojeva u nizu od N prirodnih brojeva.

■ **Zadatak 7.10**

Potrebno je izraditi dijagram toka za obradu podataka s osjetila za mjerenje mase teretnih vozila. Dostupno je N mjernih podataka mase teretnih vozila napravljenih na dionici ceste čija najveća dopuštena masa iznosi M. Potrebno je izbrojiti vozila koja su unutar dopuštene najveće vrijednosti mase, vozila koja su unutar područja tolerancije od P % te vozila čija je masa izvan dopuštenog područja. Također je potrebno naći najveću masu vozila unutar napravljenih mjerenja.

Poglavlje 8

Programski jezik C♯

Programski jezik C♯ razvio je Microsoft u sklopu .NET okvirnog programa, a prva se inačica na tržištu pojavila 2000. godine. C♯ zamišljen je kao jednostavan i suvremen objektni programski jezik koji se ubraja u skupinu viših programskih jezika. Sintaksa mu je slična s programskim jezikom C++ što mu je omogućilo brzu prihvaćenost među programerima. C♯ je strukturirani programski jezik, što znači da se naredbe izvode slijedno, naredba po naredba, od prve linije kôda glavne metode prema zadnjoj liniji kôda glavne metode. Kako programski jezik C♯ ne dopušta preskakanje linija programskog kôda, za odabir izvođenja pojedinog programskog bloka koriste se naredbe grananja ili usporedbe.

8.1 Pravila pisanja programa u programskom jeziku C♯

U programskom jeziku C♯ varijable i naredbe koje sačinjavaju programski kôd zapisuju se u tekstualne datoteke isključivo engleskim znakovima, dok se za ispisivanje obavijesti korisniku programa može koristiti i znakove koji nisu dio engleske abecede. C♯ razlikuje velika i mala slova pa je potrebno paziti na njihovu uporabu prilikom kôdiranja, odnosno prilikom rada s varijablama. Naredbe se u programskom jeziku C♯ zapisuju slijedno, odozgo prema dolje (od početka do kraja programa), na način na koji se kasnije i izvršavaju. Mjesto početka naredbe u svakom retku nije definirano, što znači da udaljenost od lijevog ruba koristimo za strukturiranje programskog kôda, a pojedinu naredbu završavamo znakom ";". U jednom retku moguće je napisati više naredbi (npr. int a; string boja = “zelena”; a = boja.length();), ali takav način zapisa često čini programski kôd teško čitljivim pa se uglavnom ne preporučuje. Programski

blokovi (odsječci) označavaju se vitičastim zagradama "{", "}", koje imaju istu funkciju kao ključne riječi "*begin*" (početak) i "*end*" (kraj) u mnogim programskim jezicima. Programski blok obuhvaća nekoliko skupina naredbi: deklaracije, definicije, druge programske blokove i pozive funkcija ili metoda (funkcije koje pripadaju nekom objektu, odnosno klasi). U programskom bloku ne moraju biti obuhvaćene sve skupine naredbi, posebno ako se radi o kratkim i jednostavnim programima. Pri deklaraciji varijabli i njihovom pozivanju potrebno je koristiti smislena imena, kako bi kôd bio lakše čitljiv i razumljiviji u slučaju da na njemu radi više osoba ili da se na njemu rade naknadne izmjene. Varijable su definirane svojim imenima pa nije dopušteno korištenje istih imena za različite varijable. Preporučeno pravilo zapisa imena varijabli je prvo malo slovo, a veliko slovo kod svake sljedeće riječi, ako se varijabla sastoji od više riječi, npr. "*prviPribrojnik*", "*brojiloStudenata*". Za imena klasa ili metoda koristi se prvo slovo veliko kao i veliko slovo u svakoj sljedećoj riječi, npr. "*SenzorGPS*", "*TeretnaKompozicija*".

Ključne riječi su unaprijed definirani identifikatori s posebnim značenjem za jezičnog prevoditelja (eng. "*compiler*"). To mogu biti: definicije varijabli, vraćanje rezultata izvođenja funkcije, operacije usporedbe, definicije petlji. Ključne riječi pišu se malim slovima i ne smiju se koristiti kao imena varijabli, funkcija i metoda. Neki primjeri ključnih riječi su: definicija tipa podatka (int, float, double, char, string, short, long), modifikatori pristupa varijabli ili metodi (private, internal, protected, public), naredbe petlji (while, for, do), naredbe grananja/usporedbe (if, else, switch, case) te naredbe upravljanja izvođenja programa (continue, break, exit). Osnovne ključne riječi programskog jezika C♯ dane su u tablici 8.1.

Općenita struktura programa napisanog u programskom jeziku C♯ sadrži: pretprocesorske naredbe (definicije konstanti, uključivanje definicija potrebnih metoda), definicije pomoćnih metoda te definiciju glavne metode (deklaracije varijabli, inicijalizacije varijabli, unos/dohvat podataka, obradu podataka te ispis podataka/rezultata).

8.2 Unos i ispis podataka

Ovisno o namjeni programa koji želimo napisati u nekom programskom jeziku, podaci koje će program koristiti može unositi sam korisnik preko neke od ulaznih jedinica ili podaci mogu biti unaprijed zapisani u neku datoteku iz koje će program čitati podatke za obradu. Osim ova dva načina unosa podataka moguć je i ponešto složeniji unos, a taj je prosljeđivanjem podataka koji su dobiveni izvođenjem nekog drugog programa, a koji nisu pohranjeni u trajnoj memoriji, na način da se podaci učitaju iz datoteke na tvrdom disku ili se dobiju preko mrežnog sučelja. Prilikom ispisa podataka kao izlazna jedinica može se koristiti

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Tablica 8.1: Ključne riječi u programskom jeziku C♯.

računalni zaslon ili pisač, a podatke se također može spremiti u datoteku koja će biti pohranjena u nekoj trajnoj memoriji. I u slučaju ispisa podataka, također se može koristiti neki oblik prosljeđivanja podataka dobivenih kao rezultat obrade u jednom programu u drugi program, bilo čitanjem pohranjene datoteke ili podataka dobivenih putem mrežnog sučelja. U slučajevima prosljeđivanja podataka mora se voditi računa o njihovoj dostupnosti nakon izvršavanja pojedinog programa ili funkcije iz skupa programa koji se koriste za neku složeniju obradu podataka.

U ovom odlomku razmotrit će se korisnički unos podataka preko konzole. Kako bi se našem programu omogućilo čitanje naredbi upisanih uporabom tipkovnice kao ulazne jedinice u prostoru pretprocesorskih naredbi potrebno je uključiti korištenje imeničkog prostora "System". U programskom kôdu tada se može koristi naredbe "*Read*" i "*ReadLine*" koje omogućavaju čitanje ulaznih podataka na sljedeći način:

"*Read*" - čita znak po znak dok operater ne pritisne tipku Enter. Znak završetka (Enter) također će biti pročitan. Metoda "*Read*" vraća vrijednost tipa int pa ako se želi da korisnikov unos bude spremljen u slovnom zapisu potrebno je koristiti pretvorbu u tip podatka char. Za pretvorbu se koristi metoda "*Convert*". Unos i pretvorba metodama "*Read*" i "*Convert*" prikazan je programskim kôdom 8.1.

Programski kôd 8.1: Učitavanje podatka i pretvorba u tip podatka char.

```
// deklaracija varijabli
char c;

// unos podataka
c = Convert.ToChar(Console.Read());
```

"*ReadLine*" - čeka da operater otipka ulazne podatke i pritisne tipku Enter. Metoda generira znak za novi redak kako bi se pokazivač na zaslonu pomaknuo na početak novog retka i vraća niz znakova osim zadnjeg znaka za novi redak (Enter). Metoda "*ReadLine*" vraća vrijednost tipa string koja se naknadno može pretvoriti u neki drugi željeni tip podatka. Primjer korištenja naredbe "*ReadLine*" i pretvorbe u tip podatka int dan je programskim kôdom 8.2.

Programski kôd 8.2: Učitavanje podatka i pretvorba u tip podatka int.

```
// deklaracija varijabli
string unos;
int cijeliBroj;

// unos podataka
unos = Console.ReadLine();
cijeliBroj = Convert.ToInt32(unos);
```

Ispis podataka na računalni zaslon kao izlaznu jedinicu može se ostvariti pomoću metoda "*Write*" i "*WriteLine*".

"*Write*" - ispisuje proslijeđeni niz znakova bez odlaska u novi redak.

"*WriteLine*" - ispisuje proslijeđeni niz znakova uz odlazak u novi redak.

Metoda "*Write*" omogućava ispis niza znakova bez automatskog ispisa znaka za novi redak. Ako se želi ispisati numeričku ili logičku vrijednost, potrebno je provesti pretvorbu numeričke ili logičke vrijednosti u niz znakova (string). Svaka varijabla ima pomoćnu metodu "*ToString*" za ostvarenje potrebne pretvorbe. Prilikom ispisa se vrijednost numeričke varijable automatski pretvara u niz znakova pa se taj dio može najčešće izostaviti. Učitavanje podatka i pretvorba u tip podatka string dani su programskim kôdom 8.3.

Programski kôd 8.3: Učitavanje podatka i pretvorba u tip podatka string.

```
// deklaracija varijabli
int cijeliBroj = 10;
string nizZnakova;
```

```
// pretvorba cjelobrojnog podatka u string
nizZnakova = cijeliBroj.ToString();
```

Ako se pak želi ispisati neki niz znakova, koji će najčešće biti poruka korisniku, to možemo učiniti stavljanjem poruke za ispis u dvostruke navodnike. Ispis niza znakova dan je programskim kôdom 8.4.

Programski kôd 8.4: Ispis niza znakova.

```
Console.Write("Niz znakova za ispis!");
```

U slučaju da se želi ispisati poruku i vrijednost varijable u istom retku, koristi se operator "+", koji omogućava spajanje više znakovnih nizova u jedan, kako je dano programskim kôdom 8.5.

Programski kôd 8.5: Ispis poruke i vrijednosti varijable.

```
// deklaracija varijable
int cijeliBroj = 10;

// ispis podataka
Console.Write("Vrijednost iznosi " + cijeliBroj.ToString());
```

Kao rezultat izvođenja ovog programskog odsječka ispis na konzoli će biti: "Vrijednost iznosi 10".

U programskom jeziku C♯ moguće i definirati format ispisa, u što se ubraja broj decimala, razmaci, postotci, novčane jedinice i sl. Neki specijalni znakovi za ispis su:

"\r" -> Return ili Enter, odnosno znak za kraj retka;

"\n" -> novi redak;

"\t" -> tab, odnosno pomak za određeni broj slovnih mjesta;

"\♯" -> znak ♯;

"\"" -> znak ";

"\\" -> znak \.

Metoda "*WriteLine*" omogućava ispis niza znakova uz automatski ispis znaka za novi redak, dok je po ostalim pravilima analogna metodi "*Write*". Ako se želi ispisati numeričku ili logičku vrijednost pohranjenu u nekoj varijabli potrebna je pretvorba numeričke i logičke vrijednosti u niz znakova. Proizvoljni niz znakova,

odnosno poruka, ispisuje se tako da se stavlja u dvostruke navodnike, a metoda "*WriteLine*" omogućava i podešavanje formata ispisa. Definiranje detalja o formatu ispisa decimalnih brojeva upisuje se unutar vitičastih zagrada, pri čemu je format definiran kao:

{redniBroj,pomak:brojCijelihMjesta.brojDecimalnihMjesta},

gdje je "*redniBroj*" varijabla čija se vrijednost želi ispisati, "*pomak*" je broj znakova od zadnje ispisanog znaka u smislu pomaka početka. Pomak može biti pozitivan (desno poravnanje) ili negativan (lijevo poravnanje). Pomak se zanemaruje ako je veći od duljine teksta za ispis. Vrijednost "*brojCijelihMjesta*" je broj znakova namijenjen ispisu znamenki lijevo od decimalnog zareza (točke u programskom jeziku C♯), a "*brojDecimalnihMjesta*" je broj znakova namijenjen ispisu znamenki desno od decimalnog zareza (točke u programskom jeziku C♯). Parametri "*brojCijelihMjesta*" i "*brojDecimalnihMjesta*" zadaju se simbolima pa primjerice "0" označava jednu znamenku. Ako znamenka ne postoji u podatku, ispiše se "0". Simbol "♯" također označava jednu znamenku. Ako znamenka ne postoji u podatku, ništa se ne ispisuje.

To se može detaljnije vidjeti na primjeru ispisa broja PI s tri decimale dan programskim kôdovima 8.6 i 8.7. Pri tome programski kôd 8.6 rezultira ispisom Broj PI: 03.142, jer su za cjelobrojne znamenke rezervirana dva mjesta. Programski kôd 8.7 rezultira ispisom Broj PI: 3.142, unatoč tome što su za cjelobrojne znamenke rezervirana dva mjesta.

Programski kôd 8.6: Ispis broja PI s tri decimale.

```
Console.WriteLine("Broj PI: {0,0:00.000}", Math.PI);
```

Programski kôd 8.7: Ispis broja PI s tri decimale bez vodeće nule.

```
Console.WriteLine("Broj PI: {0,0:##.###}", Math.PI);
```

Parametri "*brojCijelihMjesta*" i "*brojDecimalnihMjesta*" mogu se zamijeniti već predefiniranim oznakama kao što su:

P -> ispis postotka;

C -> ispis oznake novčane jedinice;

X -> heksadecimalni format;

D -> ispis cijelog broja s predznakom;

E -> ispis broja s potencijom uz 6 decimala;

N -> općeniti ispis broja s grupiranjem tisućica, drugih viših potencija i decimala;

F -> ispis datuma i vremena.

8.3 Grananja

Grananje u programu omogućava izvođenje neke naredbe ili skupa naredbi u ovisnosti o ispunjenosti nekog uvjeta. Postoji više vrsta grananja, ovisno o zahtjevima i složenosti pojedinog zadatka, a u ostatku ovog poglavlja obradit će se grananja "*if*", "*if - else*", "*if - else if - else*" i "*switch*". Ispunjenost nekog uvjeta najčešće će biti formulirana u obliku logičkog izraza, koji također rezultira logičkom vrijednosti. Ako je logička vrijednost jednaka jedinici, odnosno istinita (true), neki programski odsječak će se izvesti, a ako je logička vrijednost jednaka nuli, odnosno neistinita (false), taj programski odsječak se neće izvršiti.

8.3.1 Grananje "*if*"

Grananje uporabom naredbe "*if*" je najjednostavniji oblik grananja, pri kojem se programski odsječak izvodi ako je neki uvjet zadovoljen. Ako pak uvjet nije zadovoljen, programski odsječak se preskače i nastavlja se s naredbama koje slijede iza njega. Grananje "*if*" objasnit će se na primjeru algoritma za izračun apsolutne vrijednosti upisanog broja danog programskim odsječkom 8.8, koji je prikazan pseudokôdom 6.5 i dijagramom toka slikom 7.6.

Programski kôd 8.8: Izračun apsolutne vrijednosti.

```
// deklaracija varijabli
string unos;
int podatak;

// unos podatka
Console.Write("Unesite broj za izracun apsolutne
   vrijednosti: ");
unos = Console.ReadLine();
podatak = Convert.ToInt32(unos);

// odredjivanje radi li se o pozitivnom ili negativnom broju
if (podatak < 0)
    podatak = -podatak;

// ispis rezultata
Console.WriteLine("Apsolutna vrijednost unesenog broja
   iznosi: " + podatak);
```

8.3.2 Grananje "*if - else*"

Grananje "*if - else*" nešto je složeniji oblik grananja, pri čemu je moguće izvoditi neki programski odsječak u slučaju da uvjet nije ispunjen. To znači da će se neke naredbe moći izvoditi ako je rezultat logičkog izraza jednak nuli, bez da se nastavlja s naredbama ostatka programa. Kao primjer za uporabu grananja "*if - else*" poslužit će problem riješen pseudokôdom 6.7 i prikazan slikom 7.8, a čiji algoritam je dan programskim odsječkom 8.9. Deklarirat će se dvije potrebne varijable, jedna varijabla za spremanje praga te druga za spremanje vrijednosti koja se ispituje. Nakon unosa vrijednosti obje varijable, ispituje se je li vrijednost varijable "*podatak*" manja ili jednaka od unesenog praga. Ako je taj uvjet ispunjen ispisuje se poruka da unesena vrijednost ne prekoračuje prag i nakon toga program završava. U slučaju da uvjet nije ispunjen ispisuje se poruka da unesena vrijednost prekoračuje prag i nakon toga program završava.

Programski kôd 8.9: Provjera prekoračenja praga.

```
// deklaracija varijabli
string unos;
int podatak, prag;

// unos podataka
Console.Write("Unesite vrijednost praga: ");
unos = Console.ReadLine();
prag = Convert.ToInt32(unos);

Console.Write("Unesite vrijednost za provjeru prekoracenja
   praga: ");
unos = Console.ReadLine();
podatak = Convert.ToInt32(unos);

// odredjivanje prekoracuje li podatak zadani prag i ispis
if (podatak <= prag)
    Console.WriteLine("Unesena vrijednost ne prekoracuje
       prag.");
else
    Console.WriteLine("Unesena vrijednost prekoracuje
       prag.");
```

8.3.3 Grananje "*if - else if - else*"

Još napredniji oblik grananja je "*if - else if - else*" koji omogućava izvedbu više (najmanje tri) različita programska odsječka ovisno o rezultatima logičkih izraza. U ovom slučaju mora se definirati dva logička izraza, jedan uz naredbu

"*if*" i drugi uz naredbu "*else if*". Za primjer programskog kôda poslužit će nam primjer ranije obrađen u pseudokôdom 6.9. Na početku se deklarira varijabla "*podatak*" i unosi joj se vrijednost uz ispis pripadne poruke. Zatim se ispituje prvi uvjet kako bi ustanovili je li vrijednost varijable "*podatak*" manja od nule. Ako je ovaj prvi uvjet ispunjen, ispisuje se poruka koja operateru javlja je li unesena vrijednost manja od nule. Nakon toga program završava s radom. U slučaju da ovaj uvjet nije ispunjen program nastavlja s ispitivanjem drugog uvjeta kako bi se ustanovilo je li vrijednost varijable "*podatak*" veća od nule. Ako je sada ovaj drugi uvjet ispunjen, ispisuje se poruka koja operateru javlja je li unesena vrijednost veća od nule. Nakon toga program završava s radom. U slučaju da ni ovaj drugi uvjet nije ispunjen, izvršava se naredba vezana uz ključnu riječ "*else*" kako su svi uvjeti provjereni i niti jedan nije bio ispunjen. Ovaj slučaj je vezan za vrijednost varijable "*podatak*" jednake nuli. Ispisuje se poruka koja označava da je vrijednost varijable "*podatak*" jednaka nuli. Nakon izvršavanja ovog ispisa, program završava jer nema naredbi nakon strukture grananja.

Programski kôd 8.10: Ispitivanje predznaka broja.

```
// deklaracija varijabli
string unos;
int podatak;

// unos podatka
Console.Write("Unesite vrijednost za provjeru predznaka: ");
unos = Console.ReadLine();
podatak = Convert.ToInt32(unos);

// provjera (grananje) u slucaju da je unesena vrijednost
   negativna, pozitivna ili jednaka nuli
if (podatak < 0)
    Console.WriteLine("Unesena vrijednost je negativna.");
else if (podatak > 0)
    Console.WriteLine("Unesena vrijednost je pozitivna.");
else
    Console.WriteLine("Unesena vrijednost je jednaka
       nuli.");
```

8.3.4 Grananje "*switch*"

Grananje pomoću naredbe "*switch*" (koja se još naziva i skretnica) najsloženiji je oblik grananja. Ova naredba uvedena je kako bi se izbjegla višestruka uporaba naredbi "*if*" u slučajevima kada postoji veliki broj uvjeta pri izvršavanju programa. Naredbom "*switch*" se tako rješava problem višestrukog izbora u slučajevima kada pojedini izbor ovisi o unaprijed poznatoj vrijednosti. Važno je

naglasiti kako ovaj oblik grananja može uspoređivati samo numeričke vrijednosti.

Za ilustraciju ove vrste grananja napisan je programski kôd 8.11 koji rješava problem dan pseudokôdom 6.11. Na početku će se deklarirati varijabla "*ocjena*" i unijet će joj se vrijednost. Nakon toga se poziva skretnica vezana uz varijablu "*ocjena*". Kako ocjena može poprimiti samo jednu od pet vrijednosti iz skupa $\{1,2,3,4,5\}$ definirano je pet slučajeva. Svaki slučaj odnosi se na jednu od mogućih numeričkih vrijednosti ocjene. Program treba, ovisno od unosa, simbolički napisati unesenu ocjenu. Prilikom unosa ponekad dođe do pogreške, što znači da je potrebno osigurati sigurno izvršavanje programa i u ovom slučaju. Iz tog razloga je ključnom riječi "*default*" pridružena naredba koja ispisuje poruku operateru da nije unio valjanu numeričku vrijednost koja označava ocjenu.

Programski kôd 8.11: Ispis simboličke ocjene.

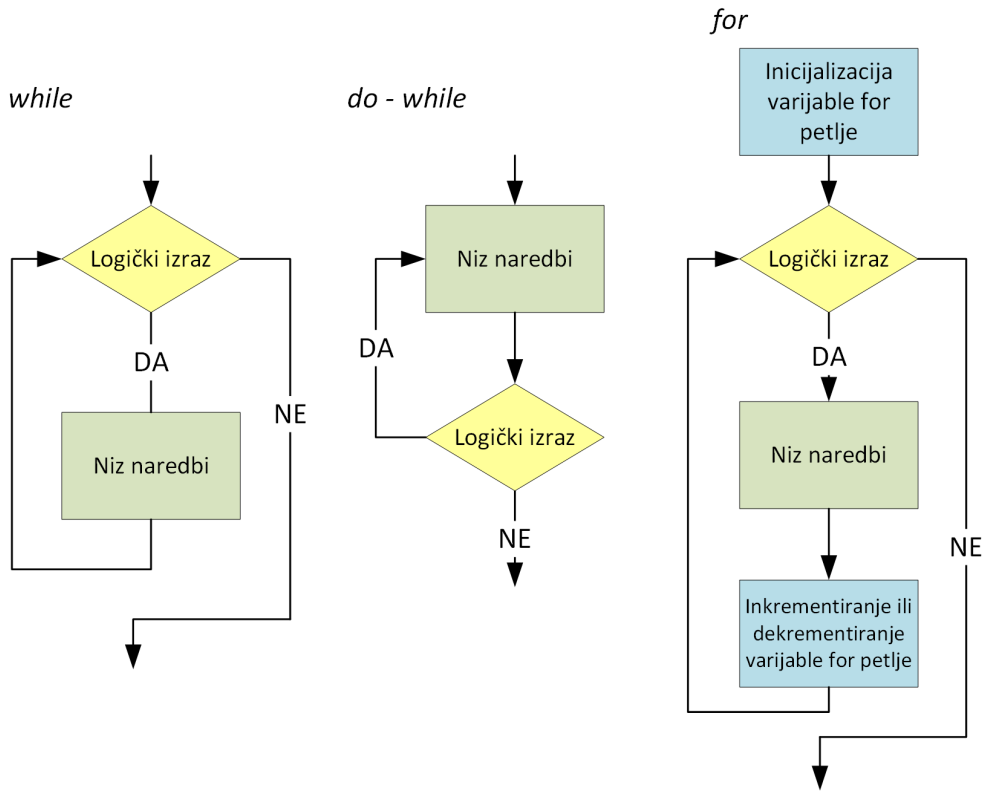
```
// deklaracija varijabli
string unos;
int ocjena;

// unos ocjene
Console.Write("Unesite numericku vrijednost ocjene: ");
unos = Console.ReadLine();
ocjena = Convert.ToInt32(unos);

// naredba grananja
switch (ocjena)
{
case 1:
        Console.WriteLine("Nedovoljan");
        break;
case 2:
        Console.WriteLine("Dovoljan");
        break;
case 3:
        Console.WriteLine("Dobar");
        break;
case 4:
        Console.WriteLine("Vrlo dobar");
        break;
case 5:
        Console.WriteLine("Odlican");
        break;
default:
        Console.WriteLine("Unesena vrijednost nije
           ocjena.");
        break;
}
```

8.4 Petlje

Petlje su vrlo često korištene naredbe posebno u slučajevima kad je potrebno neki programski odsječak koristiti više puta s različitim ulaznim parametrima. Najjednostavniji način za ostvarenje takvog zahtjeva je upravo uporabom petlji. Petlje će omogućiti ponavljanje iste skupine naredbi nad različitim ulaznim podacima dok god nije ispunjen uvjet o završetku petlje. Takvo ponavljanje iste skupine naredbi naziva se iteracija. Uvjet o završetku petlje može biti unaprijed definiran, kao broj ponavljanja izvođenja petlje ili dok neka vrijednost koja se računa unutar tijela petlje ne bude dostignuta. Pojednostavljeno se može reći kako postoje dvije osnovne vrste petlji. Prva vrsta su petlje koje imaju unaprijed definiran broj ponavljanja, jer je poznato koliko puta se želi neka operacija izvršiti ili se pak zna duljina niza elemenata nad kojima se želi neka operacija izvršiti. Druga vrsta su petlje koje se izvršavaju dok god određeni kriterij nije zadovoljen, neovisno o broju ponavljanja. Dijagrami toka za petlje "*while*", "*do - while*" i "*for*" prikazani su na slici 8.1.



Slika 8.1: Dijagrami toka za petlje "*while*", "*do - while*" i "*for*".

8.4.1 Petlja "*while*"

Petlja "*while*" pripada skupini petlji koje se izvršavaju neodređeni broj puta, tj. dok god je zadovoljen uvjet koji je postavljen. Naredbi unutar petlje može biti proizvoljno mnogo i one se izvršavaju slijedno, kao i kod svakog drugog programskog odsječka. U iteraciji kada uvjet koji je postavljen postane neistinit, odnosno bude mu pridijeljena logička vrijednost 0, naredbe sadržane unutar petlje se preskaču i nastavlja se s izvođenjem naredbi koje slijede nakon petlje. Kako bi se omogućio prekid petlje, odnosno izbjeglo beskonačno izvršavanje petlje, varijable koje utječu na uvjet petlje moraju se mijenjati u nekoj od naredbi unutar petlje. Važno je napomenuti kako se provjera ispunjenja uvjeta petlje obavlja na početku, odnosno prije početka izvođenja operacija unutar petlje. To znači da se varijable koje se koriste u uvjetu moraju inicijalizirati prije petlje i dodijeliti im se vrijednosti koje će omogućiti izvršavanje petlje. Programskim kôdom 8.12 prikazana je uporaba petlje "*while*" koja je identična onoj u primjeru obrađenom pseudokôdom 6.13 u kojem se koriste tri varijable, a program izračunava zbroj prvih "n" cijelih brojeva.

Programski kôd 8.12: Izračun zbroja prvih "n" cijelih brojeva.

```
// deklaracija varijabli i inicijalizacija
string unos;
int brojPodataka, zbroj = 0, brojiloPodataka = 1;

// unos broja pribrojnika
Console.Write("Unesite broj pribrojnika: ");
unos = Console.ReadLine();
brojPodataka = Convert.ToInt32(unos);

// petlja while kojom dobivamo rezultat zbrajanja
while (brojiloPodataka <= brojPodataka)
{
    zbroj = zbroj + brojiloPodataka;
    brojiloPodataka++;
}

// ispis rezultata
Console.WriteLine("Zbroj prvih " + brojPodataka + " cijelih
   brojeva je " + zbroj);
```

8.4.2 Petlja "*do - while*"

Petlja "*do - while*" vrlo je slična petlji "*while*" pri čemu je osnovna razlika ta da se naredbe petlje izvršavaju prije provjeravanja uvjeta, što znači da će se

naredbe ovakve petlje izvršiti barem jednom. Ako je logički izraz petlje ispunjen, one će se izvršiti i drugi put, sve dok varijable ne budu izmijenjene tako da logički izraz uvjeta petlje postane neistinit. Petlja "*do - while*" iskoristit će se na primjeru računanja umnoška prvih "n" cijelih pozitivnih brojeva, koji je dan kôdom 8.13 i pseudokôdom 6.15. Program od korisnika dobiva broj podataka, u ovom primjeru broj faktora, nakon čega računa njihov umnožak i ispisuje pripadni rezultat.

Programski kôd 8.13: Izračun umnoška prvih "*n*" cijelih pozitivnih brojeva.

```
// deklaracija varijabli i inicijalizacija
string unos;
int brojPodataka, brojiloPodataka = 1;
double umnozak = 1;

// unos broja faktora
Console.Write("Unesite broj faktora: ");
unos = Console.ReadLine();
brojPodataka = Convert.ToInt32(unos);

// petlja do-while kojom dobivamo rezultat mnozenja
do
{
    umnozak = umnozak * brojiloPodataka;
    brojiloPodataka++;
}
while (brojiloPodataka <= brojPodataka);
// ispis rezultata
Console.WriteLine("Umnozak prvih " + brojPodataka + "
   cijelih brojeva je " + umnozak);
```

8.4.3 Petlja "*for*"

Petlja "*for*" često se koristi za izvršavanje operacija kod kojih postoji unaprijed poznat broj ponavljanja tijela petlje, npr. nad elementima nekog skupa koji ima poznati broj članova. U "*for*" petlji potrebno je inicijalizirati vrijednost varijable koja se koristi kao brojač pojedine iteracije, odrediti uvjet koji ograničava izvršavanje petlje (logički izraz) i postaviti pravilo za inkrementiranje ili dekrementiranje varijable brojača. Primjer "*for*" petlje koja se izvršava dok varijabla brojača ne dostigne cjelobrojnu vrijednost jednaku 10, a započinje s vrijednosti brojača jednakoj 1 je: for (int i = 1; i < 10; i++). Osim inicijalizacije varijable unutar petlje, koja je prikazana u primjeru, varijablu brojača moguće je inicijalizirati i prije samog tijela petlje. Inicijalizacijom varijable brojača prije tijela petlje postiže se njena vidljivost nakon završetka petlja, što može biti korisno

u slučajevima kad je potrebno znati do kojeg se iznosa brojača petlja izvodila, odnosno koliko su se puta naredbe petlje izvele. Za primjer implementacije "*for*" petlje poslužit će pseudokôd 6.18 kojim je opisan izračun zbroja cijelih brojeva u nekom unaprijed definiranom intervalu, a koji je ovdje prikazan programskim kôdom 8.14. Kao ulazne podatke program dobiva gornju i donju granicu intervala, a kao izlaz nam daje zbroj svih cijelih brojeva koji se nalaze u tom intervalu s uključenim granicama intervala.

Programski kôd 8.14: Izračun zbroja cijelih brojeva unutar intervala.

```
// deklaracija varijabli i inicijalizacija
string unos;
int donjaGranica, gornjaGranica, zbroj = 0;

// unos donje granice
Console.Write("Unesite donju granicu: ");
unos = Console.ReadLine();
donjaGranica = Convert.ToInt32(unos);

// unos gornje granice
Console.Write("Unesite gornju granicu: ");
unos = Console.ReadLine();
gornjaGranica = Convert.ToInt32(unos);

// petlja for kojom dobivamo rezultat zbrajanja cijelih
   brojeva u intervalu
for (int i = donjaGranica; i <= gornjaGranica; i++)
    zbroj = zbroj + i;;

// ispis rezultata
Console.WriteLine("Zbroj cijelih brojeva u trazenom
   intervalu iznosi " + zbroj);
```

8.5 Primjeri

U ovom potpoglavlju dani su primjeri izrade programa u programskom jeziku C♯. Pri tome su iskorišteni isti primjeri dani u potpoglavlju 6.6 za objašnjenje izrade pseudokôda, odnosno potpoglavlju 7.6 za objašnjenje izrade dijagrama toka. Za izradu programa iskorištene su iste varijable tako da u nastavku neće biti ponovno navođen popis varijabli, već samo poveznica na pripadnu tablicu u potpoglavlju 6.6. Kao rješenje je dan programski kôd glavne metode kako razvojna metoda generira ostale dijelove u sklopu procedure stvaranja projekta

nakon odabira pripadnog predloška aplikacije.

■ Primjer 8.1

Napisati programski odsječak kojim se ispisuje rezultat zbrajanja dva cijela broja koje korisnik unosi preko tipkovnice. Pri ispisu pokazivač mora ostati u istom retku s rezultatom.

Rješenje:

Ovo je jednostavan problem čiji je koncept rješavanja prikazan pseudokôdom 6.2. On se može jednostavno prepisati u programski jezik C♯ čime se dobiva program 8.15. Sam program slijedi koncept dan pseudokôdom. Prvo se deklariraju potrebne varijable, a zatim se radi unos potrebnih vrijednosti. Ovdje je potrebno primijetiti kako se s konzole dobivaju vrijednosti tipa niz znakova (string) pa je potrebno napraviti pretvorbu u cjelobrojni tip podatka u sljedećem dijelu programa. Nakon pretvorbe izvršava se operacija zbrajanja nakon koje slijedi ispis rezultata uz odgovarajuću poruku operateru.

Programski kôd 8.15: Zbrajanje dva broja.

```
// deklaracija varijabli
string unos1, unos2;
int pribrojnik1, pribrojnik2, suma;

// unos podataka
Console.Write("Unesite prvi pribrojnik: ");
unos1 = Console.ReadLine();
Console.Write("Unesite drugi pribrojnik: ");
unos2 = Console.ReadLine();

// pretvorba podataka tipa niz znakova u cjelobrojni tip
pribrojnik1 = Convert.ToInt32(unos1);
pribrojnik2 = Convert.ToInt32(unos2);

// operacija zbrajanja
suma = pribrojnik1 + pribrojnik2;

// ispis rezultata
Console.Write("Zbroj dvaju brojeva iznosi " +
    suma.ToString());
```

■ Primjer 8.2

Potrebno je izraditi program koji će ispitati nalazi li se uneseni cijeli broj

u zadanom intervalu. Rezultat je potrebno ispisati u odgovarajućoj poruci operateru. Pri tome je donja granica intervala uključena, a gornja granica nije uključena u interval.

Rješenje:

Varijable potrebne za rješavanje ovog problema dane su u tablici 6.2, koncept rješenja dan je pseudokôdom 6.19 i pripadni dijagram toka prikazan je na slici 7.22. Njihovim prepisivanjem u programski jezik C♯ dobiva se program 8.16.

Program počinje deklaracijom varijabli. Kako sve varijable unosi operater nije potrebna inicijalizacija na početne vrijednosti. Nakon toga se unose potrebne vrijednosti gornje i donje granice intervala te podatka koji se ispituje. Sljedeći korak je grananje "*if - else*" kako bi se provjerilo je li uneseni podatak unutar definiranog intervala. Ovdje je potrebno primijetiti kako podatak istovremeno treba biti veći ili jednak od donje granice (ova granica je uključena) i manji od gornje granice. Iz tog razloga iskorišten je složeni uvjet provjere uz primjenu logičke funkcije I. Ovisno o tome je li uvjet ispunjen ili ne, ispisuje se pripadna poruka operateru koja označava nalazi li se podatak unutar intervala ili ne.

Programski kôd 8.16: Program za provjeru pripadnosti intervalu.

```
// deklaracija varijabli
string unos;
int podatak, donjaGranica, gornjaGranica;

// unos podataka
Console.Write("Unesite cijeli broj: ");
unos = Console.ReadLine();
podatak = Convert.ToInt32(unos);

Console.Write("Unesite donju granicu: ");
unos = Console.ReadLine();
donjaGranica = Convert.ToInt32(unos);

Console.Write("Unesite gornju granicu: ");
unos = Console.ReadLine();
gornjaGranica = Convert.ToInt32(unos);

// provjera i odgovarajuci ispis
if (podatak >= donjaGranica && podatak < gornjaGranica)
    Console.Write("Cijeli broj se nalazi u zadanom
        intervalu.");
else
    Console.Write("Cijeli broj se ne nalazi u zadanom
```

```
        intervalu.");
```

■ Primjer 8.3

Potrebno je napraviti program za izračun cestarine za sve klase vozila korištenjem skretnice. Pretpostavite da se radi o jednostavnoj dionici autoceste koja ima samo jedan ulaz i jedan izlaz. Također je potrebno izračunati iznos PDV-a te ga ispisati u odgovarajućoj poruci operateru.

Rješenje:

Programski kôd za rješenje ovo problema zasniva se na popisu varijabli u tablici 6.3 te pseudokôdu 6.20 i 6.21. Na osnovu navedenog pseudokôda može se napisati programski kôd 8.17. U ovom slučaju ne postoji dijagram toka, no pseudokôd uvijek sadrži potpuni koncept tako da je dovoljan za izradu programa u višem programskom jeziku.

U prvom dijelu programa se deklariraju potrebne varijable, a zatim slijedi unos potrebnih podataka. Unos podataka je podijeljen u dva dijela. Prvi dio se odnosi na unos PDV-a, a drugi na unos kategorije vozila. Ovdje je potrebno primijetiti kako se operateru ispisuje detaljna uputa kako unijeti ispravnu kategoriju vozila. Sljedeći dio programa odnosi se na izvršavanje skretnice koja na osnovi vrijednosti varijable "*vozilo*" pridružuje iznos cestarine varijabli "*cijena*". Pri tome se detektira i pogrešan unos kategorije vozila korištenjem opcije "*default*" u skretnici. U tom slučaju se varijabli "*cijena*" pridružuje vrijednost -1 kako bi se u sljedećem koraku umjesto ispisa cijene mogla ispisati poruka o pogrešci. Za ispis cijene se iz tog razloga koristi grananje "*if*" koje za ispravan unos kategorije vozila ispisuje cijenu cestarine te pripadni iznos PDV-a.

Programski kôd 8.17: Program za izračun cijene cestarine korištenjem skretnice.

```
// deklaracija varijabli
int vozilo;
double PDV, cijena, iznosPDVa;
string unos;

// unos podataka
Console.Write("Unesite postotak PDV-a: ");
unos = Console.ReadLine();
PDV = Convert.ToDouble(unos);

// ispis ponudjenih opcija i unos kategorije vozila
Console.WriteLine("Kategorija vozila IA -> 1");
Console.WriteLine("Kategorija vozila I -> 2");
```

```
Console.WriteLine("Kategorija vozila II -> 3");
Console.WriteLine("Kategorija vozila III -> 4");
Console.WriteLine("Kategorija vozila IV -> 5");
Console.Write("Unesite broj koji stoji uz kategoriju
   vozila: ");
unos = Console.ReadLine();
vozilo = Convert.ToInt32(unos);

// skretnica za odredivanje cijene cestarine
switch (vozilo)
{
case 1:
        cijena = 10.0;
        break;
case 2:
        cijena = 20.0;
        break;
case 3:
        cijena = 30.0;
        break;
case 4:
        cijena = 40.0;
        break;
case 5:
        cijena = 50.0;
        break;
default:
        cijena = -1.0;
        break;
}

// ispis rezultata u ovisnosti o izabranom parametru
if (cijena > 0.0)
{
    Console.WriteLine("Iznos cestarine za vozilo je: " +
       cijena + " kn.");
    iznosPDVa = cijena * PDV / 100;
    Console.WriteLine("PDV iznosi: " + iznosPDVa + " kn.");
}
else
    Console.WriteLine("Zadana je neispravna kategorija
       vozila.");
```

■ Primjer 8.4

Mnogokut (poligon) je dio ravnine omeđen zatvorenom izlomljenom linijom

ili manje formalno dio ravnine omeđen ravnim dužinama koje možemo nacrtati, a da ne podižemo olovku. Opseg mnogokuta jednak je zbroju duljina svih njegovih stranica. Potrebno je napraviti program koji će izračunati opseg mnogokuta korištenjem strukture petlje "while".

Rješenje:

Izrada potrebnog programa zasnivati će se na popisu varijabli danom u tablici 6.4, konceptu rješenja danom pseudokôdom 6.22 te dijagramu toka prikazanom na slici 7.23. Pomoću njih je moguće napisati programski kôd 8.18 zasnovan na korištenju petlje "*while*".

Kako je potrebno varijabli "*opseg*" pridružiti početnu vrijednost 0 u prvom dijelu programa se uz deklaraciju varijabli radi i njihova inicijalizacija. Zatim se unosi broj kutova mnogokuta kako bi se mogla izvršiti petlja "*while*". U tijelu petlje se prvo unese vrijednost duljine stranice, zatim se ona pridoda postojećem zbroju duljina stranice i na kraju se vrijednost kontrolne varijable poveća za 1. Nakon završetka petlje "*while*" samo se ispisuje rezultat kako varijabla "*opseg*" već sadrži opseg mnogokuta.

Programski kôd 8.18: Program za izračun opsega mnogokuta.

```
// deklaracija i inicijalizacija varijabli
int brojKutova, stranica, opseg = 0, i = 1;
string unos;

// unos podataka
Console.Write("Unesite broj kutova u mnogokutu: ");
unos = Console.ReadLine();
brojKutova = Convert.ToInt32(unos);

// izracun opsega
while (i <= brojKutova)
{
    // unos duljine stranice
    Console.Write("Unesite duljinu stranice " + i + " u
       mnogokutu: ");
    unos = Console.ReadLine();
    stranica = Convert.ToInt32(unos);

    // izracun opsega
    opseg = opseg + stranica;

    // promjena kontrolne varijable
    i++;
}
```

```
// ispis rezultata
Console.WriteLine("Opseg mnogokuta je " + opseg);
```

■ Primjer 8.5

Operacija dijeljenja je jedna od aritmetičkih operacija koja nije definirana za sve moguće vrijednosti djelitelja. Naime, dijeljenje s nulom nije definiramo što znači da je operaciju dijeljenja moguće izvršiti samo u slučaju ako je djelitelj različit od nule. Potrebno je izraditi program koji će imati ugrađenu zaštitu kod unosa podataka zasnovanu na petlji "do - while".

Rješenje:

Za izradu programskog kôda iskoristiti će se već izrađeni popis varijabli dan u tablici 6.5, koncept rješenja dan pseudokôdom 6.23 te dijagram toka prikazan na slici 7.24. Prepisivanjem u programski jezik C♯ dobiva se programski kôd 8.19.

Program započinje deklaracijom svih potrebnih varijabli. Kako sve varijable za dijeljenje unosi operater nije potrebno napraviti inicijalizaciju varijabli. U dijelu unosa podataka unos djeljenika radi se bez provjere kako ta varijabla smije poprimiti bilo koju numeričku vrijednost. Kod unosa djelitelja potrebno je spriječiti unos vrijednosti 0 i u tu svrhu je primijenjena petlja "*do - while*". U tijelu petlje se prvo unese djelitelj, zatim se grananjem "*if*" ispita njegova vrijednost radi ispisa poruke operateru. Poruka operateru je potrebna samo ako je unesena neispravna vrijednost djelitelja, odnosno vrijednost 0. Uvjet petlje "*do - while*" ispituje je li unesena vrijednost 0 za djelitelja i u tome slučaju je potrebno ponoviti unos. Nakon ispravnog unosa djelitelja, program se nastavlja izračunom rezultata dijeljenja, odnosno količnika te završava ispisom rezultata dijeljenja.

Programski kôd 8.19: Program za dijeljenje dva broja uz provjeru unosa.

```
// deklaracija varijabli
double djeljenik, djelitelj, kolicnik;
string unos;

// unos podataka dok se ne unese ispravan djelitelj
Console.Write("Unesite djeljenik: ");
unos = Console.ReadLine();
djeljenik = Convert.ToDouble(unos);

do
{
```

```
    Console.Write("Unesite djelitelj: ");
    unos = Console.ReadLine();
    djelitelj = Convert.ToDouble(unos);
    if (djelitelj == 0)
        Console.WriteLine("Djelitelj mora biti broj
           razlicit od nule!");
}
while (djelitelj == 0);

// operacija dijeljenja
kolicnik = djeljenik / djelitelj;

// ispis podataka
Console.Write("Rezultat dijeljenja s djeljenikom razlicitim
   od nule je: " + kolicnik);
```

■ Primjer 8.6

Student Fakulteta prometnih znanosti se želi natjecati za stručnu praksu u inozemstvu. Za ispunjavanje prijavnice potreban mu je prosjek ocjena iz svih do sada položenih ispita. Kako bi se studentu olakšao izračun prosječne ocjene potrebno je napraviti program koji će srednju ocjenu izračunati korištenjem petlje "for".

Rješenje:

Za izradu potrebnog programskog kôda iskoristiti će se već izrađeni popis varijabli dan u tablici 6.6, pseudokôd 6.24 te dijagram toka dan na slici 7.25. Prepisivanjem navedenog pseudokôda ili dijagrama toka u programski jezik C♯ dobiva se programski kôd 8.20.

U prvom dijelu dobivenog programskog kôda deklariraju se sve potrebne varijable pri čemu se varijabla "*zbrojOcjena*" inicijalizira na početnu vrijednost 0. Zatim slijedi unos količine ocjena čime su skupljeni svi podaci potrebni za realizaciju petlje "*for*". U tijelu petlje "*for*" se prvo unosi ocjena, a zatim se ona pridoda trenutnoj vrijednosti zbroja ocjena. Ovdje je dobro primijetiti da petlja "*for*" koristi lokalnu varijablu brojača "*i*" koja je deklarirana u sklopu koraka inicijalizacije petlje "*for*". Nakon što petlja "*for*" završi, varijabla "*zbrojOcjena*" sadrži zbroj svih ocjena studenta i moguće je izračunati traženu vrijednost tako da se ta varijabla podijeli brojem ocjena. Prilikom računanja srednje vrijednosti ocjena potrebno je obratiti pažnju da su varijable "*zbrojOcjena*" i "*brojOcjena*" cjelobrojne. Njihovim dijeljenjem se ponovno dobiva cijeli broj što ne daje pravu informaciju o prosječnoj ocjeni. Iz tog razloga je potrebno tip obje

varijable pretvoriti u broj s plivajućim zarezom (tip float ili double) korištenjem ukalupljivanja (eng. *cast*). Na kraju programa se ispisuje izračunata srednja ocjena.

Programski kôd 8.20: Program za izračun srednje ocjene.

```
// deklaracija i inicijalizacija varijabli
int ocjena, brojOcjena, zbrojOcjena = 0;
double srednjaOcjena;
string unos;

// unos podataka
Console.Write("Unesite broj ocjena: ");
unos = Console.ReadLine();
brojOcjena = Convert.ToInt32(unos);

// unos ocjena i izracun zbroja ocjena
for (int i = 1; i <= brojOcjena; i++)
{
    Console.Write("Unesite ocjenu: ");
    unos = Console.ReadLine();
    ocjena = Convert.ToInt32(unos);
    zbrojOcjena = zbrojOcjena + ocjena;
}

// racunjanje srednje ocjene
srednjaOcjena = (double) zbrojOcjena / (double) brojOcjena;

// ispis podataka
Console.Write("Srednja ocjena studenta je: " +
   srednjaOcjena);
```

■ Primjer 8.7

Napisati programski odsječak koji ispisuje rezultat zbroja svih unesenih brojeva dok korisnik ne unese broj 0.

Rješenje:

Proučavanjem zadanog problema moguće je zaključiti kako su za rješenje potrebne tri varijable. Jedna pomoćna varijabla za prihvat niza znakova s konzole, varijabla za unos pojedinog broja te varijabla za spremanje zbroja unesenih brojeva. Kako u opisu problema nije točno napomenut tip podatka pretpostavlja se kako se radi o cijelim brojevima. Na osnovi toga moguće je sada kreirati popis varijabli dan u tablici 8.2.

Ime varijable	Tip varijable	Značenje varijable
unos	niz znakova	Pomoćna varijabla za unos
pribrojnik	cijeli broj	Uneseni podatak
suma	cijeli broj	Zbroj unesenih brojeva

Tablica 8.2: Popis varijabli za izračun zbroja unesenih brojeva dok korisnik ne unese broj 0.

U opisu zadanog problema spominje se obrada veće i nepoznate količine brojeva. To automatski upućuje da je potrebno iskoristiti neku od petlji. Za prijedlog rješenja iskoristi će se petlja "*do - while*", a za vježbu moguće je napraviti rješenje korištenjem petlje "*while*" ili "*for*". Kod petlje je bitno definirati uvjet završetka. Kako unos treba završiti kada se unese broj 0, tijelo petlje se prema tome mora izvršavati dok god je unesena vrijednost različita od 0. Budući da je potrebno unijeti više podataka i kako se unosi podatak po podatak, unos brojeva za izračun sume će se raditi unutar tijela petlje. Za izračun zbroja iskoristiti će se zakonitost određivanja zbroja brojeva korištenjem petlje opisana u potpoglavlju 6.5.

Na osnovi iznesenoga moguće je sada napisati programski kôd 8.21. Na početku programa deklariraju se potrebne varijable. Istovremeno se varijabla "*suma*" postavlja na početnu vrijednost 0. Kako količina podataka za obradu nije unaprijed poznata, unos i obrada su spojeni u jedan dio u sklopu tijela petlje "*do - while*". U tijelu petlje se prvo unese novi podatak, zatim se on pridoda trenutnom zbroju do sada unesenih brojeva te se nakon toga provjerava uvjet za nastavak petlje. Petlja nastavlja s izvršavanjem dok god je uneseni podatak različit od nule. Nakon završetka petlje ispisuje se rezultat, odnosno izračunata vrijednost zbroja unesenih brojeva.

Programski kôd 8.21: Program za izračun zbroja brojeva dok operater ne unese broj 0.

```
// deklaracija varijabli
string unos;
int pribrojnik, suma = 0;

// unos podataka dok se ne upise 0 i izracun zbroja
do
{
    Console.Write("Unesite pribrojnik, za kraj 0: ");
    unos = Console.ReadLine();
    pribrojnik = Convert.ToInt32(unos);
    suma = suma + pribrojnik;
}
while (pribrojnik != 0);
```

```
// ispis rezultata
Console.Write("Zbroj unesenih brojeva iznosi " +
    suma.ToString());
```

■ **Primjer 8.8**

Napisati program koji će čitati pozitivne brojeve s tipkovnice dok se ne upiše broj nula i zatim ispisati najveći broj. Zanemariti negativne brojeve.

Rješenje:

Iz opisa problema moguće je zaključiti kako su za kreiranje programa potrebne najmanje dvije varijable ukoliko se koristi ulančavanje jednostavnijih naredbu u složenije kod unosa podataka. U slučaju ulančavanja kod unosa nije potrebno deklarirati pomoćnu varijablu tipa niz znakova. Jedna varijabla je potrebna za unos podatka za obradu, a druga varijabla je potrebna za spremanje trenutne najveće vrijednosti. Iz ovoga je sada moguće napraviti popis varijabli dan tablicom 8.3.

Ime varijable	Tip varijable	Značenje varijable
broj	cijeli broj	Uneseni podatak
najveci	cijeli broj	Pronađeni najveći broj

Tablica 8.3: Popis varijabli za najvećeg pozitivnog broja dok korisnik ne unese broj 0.

Kako se ponovno radi o obradi veće količine podataka iskoristiti će se struktura petlje, odnosno ponovno petlja "*do - while*". Za vježbu je moguće napraviti rješenje korištenjem petlje "*while*" ili "*for*". Kod određivanja najvećeg broja potrebno je pažnju obratiti na početnu vrijednost varijable u koju se sprema najveći broj. U ovom slučaju zadaje se kao početna vrijednost varijable "*najveci*" vrijednost 0. Kako se zanemaruju negativne vrijednost, svaka valjana unesena vrijednost će biti jednaka ili veća od zadane početne vrijednosti. Ponovno je u sklopu tijela petlje potrebno prvo napraviti unos nove vrijednosti te provjeriti je li unesena nova najveća vrijednost. Pri tome je u koraku provjere pomoću grananja "*if*" potrebno uzeti u obzir i provjeru je li uneseni broj valjan, odnosno veći od 0. Tijelo petlje će se ponovno izvršiti ako je uneseni broj različit od 0 kako samo zanemarujemo negativne brojeve. Nakon što petlja završi ispisuje se rezultat. Prilikom ispisa rezultata radi se ponovno provjera za slučaj da operater nije unio broj veći od 0 što znači da nije bilo valjanih podataka za obradu. U tu svrhu je iskorišteno grananje "*if*".

Programski kôd 8.22: Program za određivanje najvećeg pozitivnog broja dok operater ne unese broj 0.

```
// deklaracija i inicijalizacija varijabli
int broj, najveci;
najveci = 0;

// unos podataka dok se ne upise 0
do
{
    Console.Write("Unesite broj (0 za kraj) > ");
    broj = Convert.ToInt32(Console.ReadLine());
    if (broj > 0 && broj > najveci)
            najveci = broj;
}
while (broj != 0);

// ispis rezultata uz navedene uvjete koji se provjeravaju
   grananjem if
if (najveci > 0)
    Console.WriteLine("\nNajveci ucitani broj je > " +
       najveci);
else
    Console.WriteLine("\nNije ucitan broj veci od 0!");

System.Console.ReadKey();  // omogucava da ispis ostane
   vidljiv do pritiska bilo koje tipke na tipkovnici
```

8.6 Zadaci za samostalan rad

Za samostalan rad se mogu koristiti zadaci iz prethodnih poglavlja 6 i 7. Dobru podlogu za daljnje vježbanje čini i knjiga [16] u kojoj uz teorijsko objašnjenje programskog jezika C♯ postoje i riješeni zadaci te pitanja za samostalan rad. Manja količina zadataka vezana za pretvorbu brojeva te izradu dijagrama toka nalazi se i u knjizi [13].

Bibliografija

[1] Donald Knuth. Algorithms in modern mathematics and computer science. Technical Report STAN-CS-80-786, Stanford University Computer Science Department, Stanford, SAD, Siječanj 1980.

[2] Terry Wilson, Martin C. Carlisle, Jeff Humphries i Jason Moore. RAPTOR - Flowchart Interpreter. `raptor.martincarlisle.com/`, 2003. [pristupljeno 08. travnja 2015.].

[3] Visual Studio - Microsoft Developer Tools. `https://www.visualstudio.com/`, 1995. [pristupljeno 08. travnja 2015.].

[4] Xuan Li. No Excuse List. `www.noexcuselist.com/`, 2012. [pristupljeno 08. travnja 2015.].

[5] Darko Grundler. *Osobna računala.* INA - industrija nafte, INFO Zagreb, 1994.

[6] John von Neuman. First draft of a report on the EDVAC. Technical Report W-670-ORD-4926, Moore Institute of Electrical Engineering, University of Pennsylvania, 1945.

[7] Željko Drakulić, Andrej Jukić, Chloé Onsea, Igor Orešković i Natalia Reinić. Građa računala. `ahyco.uniri.hr/Seminari2008/Gradja_racunala/osnovni.html`, 2008. [pristupljeno 13. travnja 2015.].

[8] J. Glenn Brookshear. *Computer Science - An Overview.* PEARSON Addison Wesley, 2009.

[9] Danko Basch i Mario Kovač. *Osnove procesora FRISC.* Antonić d.o.o., 2004.

[10] Darko Grundler. *Primijenjeno računalstvo.* Graphis d.o.o., Maksimirska 88, Zagreb, 1998.

[11] Milan Korać i Dario Car. *Uvod u računalne mreže*. Algebra d.o.o., Zagreb, 2014.

[12] Edouard Ivanjko. Autorizirana predavanja iz predmeta Računalstvo. `www.fpz.unizg.hr/eivanjko/courses/racunalstvo.html`, 2012. [pristupljeno 25. kolovoza 2015.].

[13] Hrvoje Gold, Dražen Kovačević i Zvonko Kavran. *Informatika u prometnom inženjerstvu*. Fakultet prometnih znanosti Sveučilište u Zagrebu, Vukelićeva 4, Zagreb, 2005.

[14] American Society of Mechanical Engineers. ASME standard; operation and flow process charts. Technical report, The American Society of Mechanical Engineers, New York, 1947.

[15] Terry Wilson, Martin C. Carlisle, Jeff Humphries i Jason Moore. RAPTOR: introducing programming to non-majors with flowcharts. *Journal of Computing Sciences in Colleges*, 19(4):52–60, Siječanj 2004.

[16] Milan Gocić. *Programski jezik C♯: Pitanja, odgovori i rešeni zadaci*. Mikro knjiga, 2013.