Izrada dijagrama toka

Priču o algoritmima započeti ćemo onako kako počinju sve klasične priče. Nekada davno živio je u Bagdadu pisac, matematičar, astronom i geograf po imenu **Muhammed ibn Musa al Khowarizmi**. Vjerojatno nije bio niukakvoj vezi s Ali Babom i 40 hajduka. Isto tako vjerojatno nije niti sanjao tamo daleke 852. godine kada je pisao knjigu ***Kitab al jabr w'al-muqubala*** da će od toga nastati čak dva uzroka glavobolje đacima deset-jedanaest stoljeća nakon toga.

Njegov ***al jabr*** je postala **algebra**. Isto tako pravila rješavanja iz početka prvenstveno matematički problema su iskrivljavanjem imena **al Khowarizmi** nazivana **algoritmi**. No, tu greškama nije kraj. Prema korijenu riječi bilo bi pravilno umjesto algoritmi koristiti **algorizmi**. Ali kako jezik nije znanost nego skup dogovora, izgleda da je dogovoreno da najčešće greške postaju jezična pravila.

Tako mi danas imamo algoritme, a oni su niz preciznih uputa koje nas korak po korak vode do rješenja nekog problema. To su zapravo toliko precizne upute da za njihovo izvršavanje nije potrebna inteligencija. Zašto bi se mi bavili stvarima za koje ne treba nimalo pameti? Zato što izrada algoritma u programiranju prethodi samom pisanju programa. Pri pisanju programa prvo nam mora biti jasno što se zapravo od programa očekuje. Kao i kod rješavanja zadataka u bilo kom području prvo moramo znati postaviti problem. Drugi korak je gruba skica rješenja. Treći korak je izrada algoritma. Ako smo dobro napravili algoritam, tada pisanje programa nije ništa drugo doli prepisivanje algoritma u neki programski jezik. Kada smo to učinili dobili smo program, a program "tjera" računalo da radi ono što mi zapravo želimo. Netko je rekao da je računalo idiot velike brzine. To je ustvari bit priče o algoritmima. Mi put do rješenja moramo toliko rascjepkati i detaljno napisati da bi ga mogao razumjeti i taj "idiot velike brzine". Drugim riječima, upute moraju biti jednostavne i precizne tako da ih može izvršavati i stroj.

Autori udžbenika vole algoritme objašnjavati na receptima za pripravu nekih jela, pa je tako u udžbenicima za informatiku toliko recepata da bi se mogla izdati posebna kuharica autora informatičkih udžbenika. Pošto nekako sumnjam da su informatičari dobri kuhari, mi ćemo probati pronaći neke druge primjere. Pokušajte se sjetiti gdje ste zadnji put pročitali neko detaljno uputstvo. Možda vam je i bilo u rukama, ali niste čitali jer to već znate napamet, kao npr. obnoviti račun mobitela putem bona. Ili možda da probamo obrnuto. Da li ste se nedavno našli u situaciji da niste nešto znali napraviti jer su upute bile toliko loše i nerazumljive ili uopće niste imali upute?

U svakodnevnom životu smo zapravo stalno u doticaju s algoritmima, a često i postupamo po algoritmima da toga nismo niti svjesni. Znati "algoritamski" razmišljati dobro je bez obzira bavili se vi poslije programiranjem ili ne.

Koliko je 2 i 2

Gornji naslov jedno je od najčešćih pitanja koje se upućuje predškolskom djetetu kada ga iskušavate u računanju. Isto tako postoji uzrečica da je nešto "jasno kao dva i dva". No, da li je baš sve tako jasno? U prethodnoj lekciji smo rekli da upute u algoritmu moraju biti tako jasne da se mogu izvršavati i do rješenja doći bez imalo inteligencije. Kad već nema nimalo inteligencije morati ćemo se poslužiti džepnim računalom. Ništa jednostavnije nego otipkati "dva i dva" i dobiti rješenje. Evo nam prvog problema. Na džepnom računalu nema "i". Mi podrazumijevamo da je "dva i dva" zapravo dva više dva, da se radio o matematičkoj operaciji zbrajanja. Džepno računalo to ne zna. Znači moramo biti precizniji i reći 2+2. Baš smo riješili prvi problem kad se pojavio i drugi. Tipkam 2+2 i ništa se ne događa. I bilo je za očekivati jer je džepno računalo isključeno. Nitko nije rekao da prvo provjerim da li je džepno računalo uključeno. Sada sam već pomalo nervozan što se tako jednostavna stvar toliko komplicira. Uključio sam računalo i tipkam 2+ upsss... Ovako iznerviran htio sam što prije riješiti zadatak i u žurbi sam prstom zakvačio i tipku iznad 2 tako da je ispalo 52. Što ću sad? Jednostavno ću pritisnuti tipku CE koja poništava zadnji unos i ponovno upisati 2. Hajdemo ovo sve što smo zapetljali pokušati prepisati u čistopis u obliku natuknica:

- provjerite da li je džepno računalo uključeno
  - ako nije uključite ga
- upišite 2
  - ako ste pogriješili pritisnite tipku CE i ponovite radnju iz prethodne natuknice

- upišite +
  - ako ste pogriješili pritisnite tipku CE i ponovite radnju iz prethodne natuknice
- upišite 2
  - ako ste pogriješili pritisnite tipku CE i ponovite radnju iz prethodne natuknice
- pritisnite tipku =
  - ako ste pogriješili ponovite sve radnje počevši od trećeg reda
- pročitajte rezultat
- isključite džepno računalo

Mislite da je tu kraj? Stvari se mogu još zakomplicirati. Što ako je to model džepnog računala bez tipke CE koja poništava zadnji unos? Pokušajte doraditi gornje upute za takav model džepnog računala. Je li sada konačno kraj? Možemo li isključiti džepno računalo? Ne. Džepno računalo koje imam ispred sebe ima sunčanu ćeliju i ne gasi se na tipku. Jednostavno se samo isključi ako se neko vrijeme ne koristi. Možda ovo izgleda kao da vas gnjavim, ali s računalom je tako. Ponekad stvarno izgleda kao da se malo pravi ludo. Treba se sjetiti da je računalo "idiot velike brzine" i da ništa ne zna samo. Ako nešto "zna" napraviti onda mu je to netko morao "reći" i detaljno objasniti. Isto tako ako nešto krivo radi - samo radi ono što smo mu naredili.
Razmislite o ovome prije nego što ponovo za nešto kažete da je jednostavno kao "dva i dva".

Pseudokod

Algoritmi kojima smo se bavili u prošlim lekcijama mogu se zapisati na više načina, a dva najčešća su **pseudokod** i **dijagram toka**.

Pseudokod je algoritam zapisan riječima. ***Pseudo*** nam dolazi od grčkog laž, lažni, nadri, nazovi, tobožnji... ***Kod*** je skup dogovorenih znakova kojima možemo oblikovati neku poruku, komunicirati. U programiranju se pod kodom razumijeva izvorni program napisan u nekom programskom jeziku. Prema tome, naš pseudokod bi bio program "lažnjak". Zašto pseudo? Jer upute ne pišemo u nekom od programskih jezika već dogovorenim riječima i znakovima iz govornog jezika.

Dakle, ako hoćemo pisati algoritme riječima moramo se prije dogovoriti koje ćemo riječi koristiti. Logičan slijed radnji u nekom programu je ulaz podataka, obrada i izlaz podataka. Prema tome ćemo i odabrati prikladne riječi koje ćemo opisati u tablici:

početak	svaki algoritam ima samo jedan početak
upiši ili ulaz ili unesi ili učitaj	ovo bi bile prikladne riječi za unos podataka preko tipkovnice, miša, bar-kod čitača ili neke druge ulazne jedinice
ispiši ili izlaz	ispis podataka na zaslonu monitora, pisaču...
kraj	broj instrukcija u algoritmu mora biti konačan, svaki algoritam mora imati kraj i to samo jedan kraj

U tablici nema riječi za obradu podataka jer za to obično koristimo izraze koji nisu ništa drugo doli prepisane n pr. matematičke formule. Idemo probati napisati pseudokod za naš "dva i dva" iz prethodne lekcije. No, više nećemo koristiti džepno računalo nego ćemo algoritam napisati kao da pišemo program, odnosno upute pravom računalu:

```
početak
upiši 2
```

Ovdje ćemo malo zastati. Ako bismo napisali upiši 2 tada bi naš algoritam bio samo za zbrajanje 2 i 2 i cijeli svoj život bi radio samo to. Računalni se programi pišu što općenitije, pa ćemo i mi pokušati napisati algoritam za zbrajanje bilo koja dva broja. Kako ne možemo predvidjeti koja će dva broja netko poželjeti zbrojiti, morati ćemo umjesto brojeva koristiti varijable. U matematici je varijabla neka promjenjiva veličina, a u našem programu to će biti mjesto u radnoj memoriji koje je rezervirano za broj koji zbrajamo (ne bi bilo loše ovdje se malo podsjetiti von Neumannovog modela računala). U našem primjeru će nam trebati tri takova mjesta. Neka to budu varijable **a** i **b** za brojeve koje zbrajamo i **c** za rezultat. Pokušajmo sada ponovno napisati pseudokod:

```
početak
upiši a, b
```

```
c = a + b
ispiši c
kraj
```

U trećem redu našeg pseudokoda izraz za zbrajanje varijabli a i b napisali smo malo naopako od onoga kako bismo dva broja zbrojili na papiru. Isto tako taj treći red ne bismo pročitali kao u matematici c je jednako a + b. Sjetimo se da radimo s mjestima rezerviranim u radnoj memoriji. Tako sadržaj mjesta u memoriji za rezultat c postaje jednak zbroju sadržaja mjesta u kojima su spremljeni brojevi a i b. Zato to i čitamo c postaje a + b.

Za vježbu: Po uzoru na pseudokod za zbrajanje dva broja probajte napisati algoritme za jednostavne probleme kao što su izračun površine pravokutnika, pravokutnog trokuta, opseg i površinu kruga, brzinu ako su poznati put i vrijeme, silu ako su poznati masa i akceleracija...

Varijable i operatori

Pri razmatranju pseudokoda pokušali smo objasniti što su varijable. Prisjetimo se da se podaci i instrukcije (program) nalaze u radnoj memoriji računala (RAM). Procesor izvršava instrukcije programa jednu po jednu. Na osnovu instrukcija uzima podatke iz memorije, obrađuje ih, te rezultate obrade vraća nazad u memoriju na za to predviđena mjesta i odatle ih šalje na neku od izlaznih jedinica - monitor, pisač, zvučnike...

Za programiranje je bitno shvatiti postupanje s varijablama. Ovisno od programskog jezika u kojem ćemo pisati programe, podatke koji će biti smješteni u pojedine varijable treba više ili manje detaljno opisati. Rekli smo već da se pojedini podaci smještaju u rezervirana mjesta u memoriji. Ti dijelovi memorije imaju svoju adresu i mi im u postupku programiranja dajemo imena. Radi racionalnog korištenja memorije varijable treba definirati, tj. reći računalu koji tip podataka će biti smješten u kojoj varijabli. To mogu biti cjelobrojne vrijednosti, realni brojevi, logičke vrijednosti, jedan znak ili niz znakova...

Mi ćemo se uglavnom baviti problemima s nekim brojčanim vrijednostima. Za različite operacije nad podacima koristimo različite **operatore**. Operatori koje ćemo koristiti malo su drugačiji od onih na koje smo navikli dok "pješice" radimo npr. matematiku.

Operacija	Operator
zbrajanje	+
oduzimanje	-
množenje	*
dijeljenje	/

U gornjoj tablici su dani operatori za osnovne računske operacije. U nekim programskim jezicima operatori za operacije nad realnim i nad cijelim brojevima nisu isti. Tako npr. se za dijeljenje cijelih brojeva može koristiti operator **div** koji kao rezultati daje opet cijeli broj.

Pogledajmo tablicu s komentarima:

11.9394 / 2.2 = 5.427	dijeljenje dva broja operatorom za realne brojeve daje rezultat realan broj
11 / 2 = 5.5	
10 / 2 = 5.0	realan broj će biti i rezultat dijeljenja dva cijela broja ako upotrijebimo operator dijeljenja za realne brojeve, zato rezultat pišemo s jednim decimalnim mjestom ako nije drugačije određeno
10 div 2 = 5	ako upotrijebimo operator dijeljenja cijelih brojeva rezultat će biti cijeli broj čak i onda kada djeljenik nije djeljiv s djeliteljem
11 div 2 = 5	

Kod dijeljenja cijelih brojeva pojavljuje se još jedan koristan operator. To je **mod**. Rezultat dijeljenja cijelih brojeva s operatorom mod daje ostatak cjelobrojnog dijeljenja. Pogledajmo primjere:

11 mod 2 = 1	11 podijeljeno s 2 je 5 i **1** je ostatak

10 mod 3 = 1	10 podijeljeno s 3 je 3 i **1** je ostatak
8 mod 3 = 2	8 podijeljeno s 3 je 2 i **2** je ostatak
8 mod 2 = 0	8 podijeljeno s 2 je 4 i **0** je ostatak

Zašto je operator mod koristan. Na primjer, pomoću njega možemo utvrditi da li je neki broj djeljiv s nekim drugim brojem. Ako je, ostatak takovog dijeljenja mora biti 0. Ili npr. da li je neki broj smješten u određenoj varijabli u memoriji paran ili neparan. Paran broj dijeljen s 2 daje ostatak 0, odnosno neparan broj dijeljen s dva daje 1 ostatak.

Pokušajte doći do rezultata slijedećih izraza:

A = 5; B = 7; C = A MOD 3 + B	C = _____
A = 3; B = 2; R = A DIV B	R = _____
A = 7; B = 4; X = 2 * B / A	X = _____

Isto tako ponekad će trebati usporediti neke veličine. Zato koristimo **relacijske operatore**:

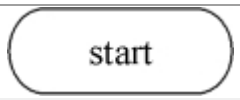
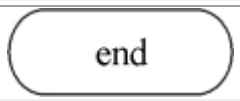
Operator	Opis
>	veće
<	manje
=	jednako
>=	veće ili jednako
<=	manje ili jednako
<> ili #	različito

Dijagram toka

Rekli smo već da se algoritmi najčešće zapisuju kao **pseudokod** i **dijagram toka**.

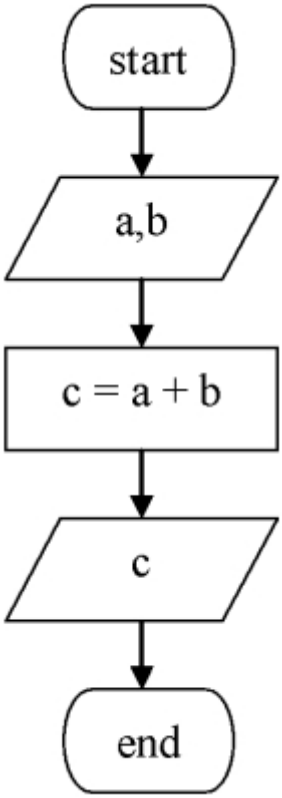
Dijagram toka je grafički prikaz algoritma. Takav način zapisivanja ima nekoliko prednosti pred pseudokodom. Zapisivanje se vrši **međunarodno dogovorenim simbolima** i ne ovisi o govornom jeziku onoga koji sastavlja algoritam. Grafički prikaz je jednostavan, pregledan, lako se pronalaze greške. Nadalje, problem se može jednostavno analizirati, usporediti s nekim drugim problemom, skratiti vrijeme pronalaženja rješenja.

Evo osnovnih simbola dijagrama toka:

	početak
	ulaz podataka
	deklaracija varijabli i konstanti; postavljanje na početnu vrijednost; obrada podataka
	izlaz podataka
	kraj
	spojna točka, radi lakšeg praćenja toka podataka obično se u spojnu točku upisuju brojevi koji su veze između različitih dijelova algoritma

Simboli za ulaz i izlaz podataka su istog oblika. Kod jednostavnih algoritama ulaz i izlaz su odmah uočljivi. Međutim, kod vrlo složenih algoritama dobro je u paralelogram koji simbolizira ulaz

pri dnu povući vodoravnu crtu i ispod nje napisati riječ ulaz. Isto to je dobro učiniti i za izlaz podataka. Simboli dijagrama toka se povezuju strelicama koje pokazuju tok podataka.



Evo ponovno našeg "**dva i dva**".

Nakon početka prvo će netko morati preko tipkovnice upisati dva broja koja želi zbrojiti. Zamislimo 3 i 5.
Ta dva broja smještaju se u radnoj memoriji kao varijable **a** i **b**. Drugim riječima broj 3 čuči u dijelu memorije koji smo nazvali **a**, broj 5 se raskomotio u dijelu memorije koji smo nazvali **b**.

Sadržaj varijable **c** postaje zbroj sadržaja varijabli **a** i **b**. Odnosno u dio memorije koji smo nazvali c smješta se zbroj brojeva 3 i 5, dakle 8.

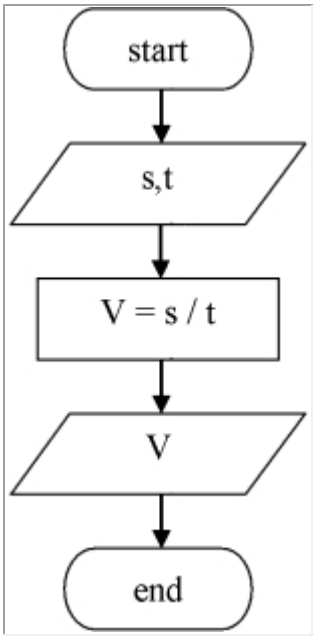
Krajnji cilj obrade je dobiti rezultat.
Ispisujemo sadržaj varijable **c**. Na zaslonu monitora ili na pisaču, ovisno kako jednoga dana budemo to definirali u programu, pojaviti će se broj 8.

Kraj.

Slijed

Dijagram toka kakav smo razmatrali za zbrajanje dva zbroja često se naziva slijed jer naredbe slijede pravocrtno jedna iza druge. Stariji programeri znaju reći da program "propada" do kraja. Idemo za vježbu napraviti još jedna slijed.

Iz osnovne škole sjećamo se da je brzina nekog tijela zapravo prijeđeni put u jedinici vremena. Matematički bismo to zapisali kao **V = s / t**, gdje je **V** brzina, **s** prijeđeni put i **t** vrijeme. To je tako jednostavno da nam nije potrebna nikakva priprema i gruba skica rješenja. Možemo odmah raditi algoritam.



Izmjereni put i vrijeme treba upisati u računalo.

Našu formulu iz zadataka samo ćemo prepisati kao uputu što treba učiniti s varijablama **s** i **t**. Sadržaj varijable **V** postaje rezultat dijeljenja sadržaja varijable **s** sa sadržajem varijable **t**.

Ispisujemo rezultat, odnosno našu izračunanu brzinu **V**.

Kraj.

Za vježbu: Pokušajte prema ovo primjeru nacrtati dijagram toka za izračun površine pravokutnika, a zatim dijagram za izračun opsega pravokutnika.

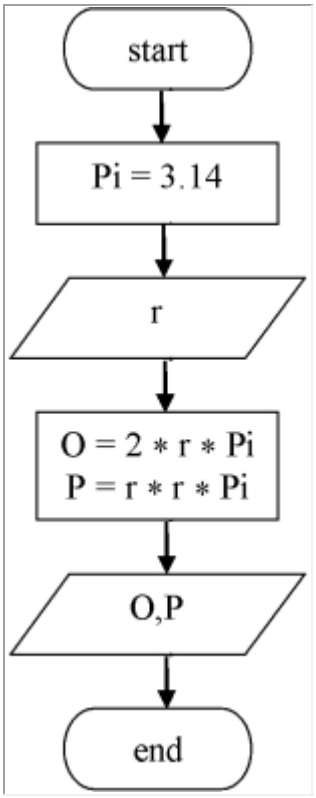
Evo još jednog primjera za izračunati opseg i površinu kruga. Kako na tipkovnici nemamo na raspolaganju grčka slova, našu formulu za izračun opsega pisati ćemo kao

$$O = 2 * r * Pi$$

Isto tako za sada još ne znamo funkciju za kvadriranje pa ćemo formulu za površinu kruga pisati ovako:

$$P = r * r * Pi$$

gdje je r * r zapravo r^2.



Za razliku od primjera koje smo radili do sada ovdje se prvi puta pojavljuje jedna konstanta. Broj Pi ćemo mi zapisati u programu zato što je on uvijek isti i sasvim suvišno bi bilo da ga netko upisuje putem tipkovnice svaki put kada se pokrene program. Isto tako izbjegli smo mogućnost da netko krivo upiše podatak za broj Pi.

Upisujemo polumjer kruga i on se smješta u varijablu **r**.

Izračunavamo opseg i površinu.

Ispisujemo rezultate.

Kraj.

Za vježbu: Po uzoru na opseg i površinu kruga, pokušajte algoritme za izračun opsega i površine pravokutnika spojiti tako da u jednom algoritmu bude i opseg i površina pravokutnika.

Grananje

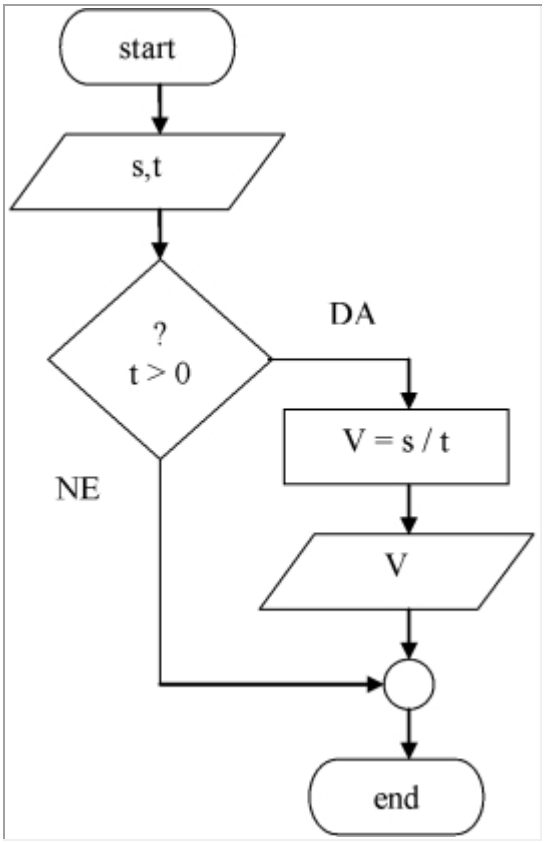
Na žalost, u stvarnom životu zbivanja ne teku tako jednostavno kao što su to pravocrtni algoritmi. Ponekad treba stati, razmisliti, usporediti neke stvari i donijeti odluku kako dalje. Takovo mjesto u algoritmu zove se grananje.

Simbol za grananje je romb. Obično se u gornji vrh upiše upitnik i u sredini se postavi logički uvjet. Grananje uvijek ima jedan ulaz i najmanje dva izlaza. Ako su samo dva izlaza tada je riječ o jednostrukom grananju, a odgovori na postavljeni logički uvjet mogu biti **DA** i **NE**, odnosno **T** i **F** (true i false). Ako je moguće više odgovora tada je to višestruko grananje kojim se u ovom tečaju nećemo baviti jer nam dolazi po programu u drugom razredu.

Sjetimo se zadataka iz prethodne lekcije s izračunom brzine ako su poznati put i vrijeme:

V = s / t.

Ako se na računalu dogodi situacija da dijelimo s nulom program će nam izbaciti pogrešku. Da bismo spriječili "ispadanje" programa, prije izračuna brzine provjeriti ćemo da li je upisano vrijeme veće od 0.



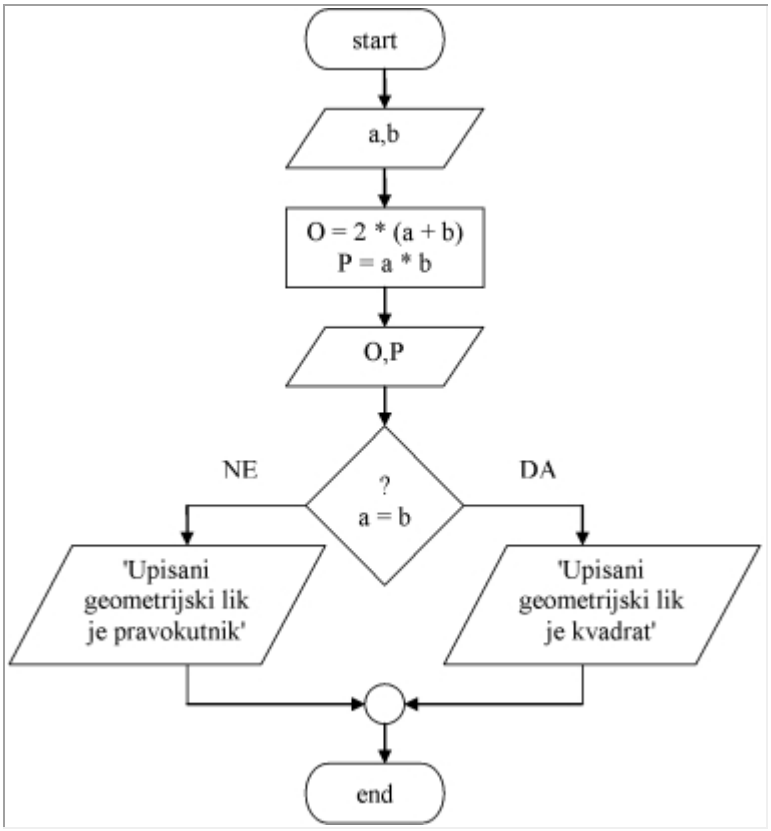
Kao i u prethodnom zadatku s brzinom, prvo upisujemo neke vrijednosti za prijeđeni put **s** i vrijeme **t**.

Ako je vrijeme **t** veće od **0** tada se izvršavaju naredbe na grani **DA**, izračunava se brzina i ispisuje rezultat.

Ako upisano vrijeme **t** nije veće od **0** tada se ne događa ništa, program odlazi na kraj i završava s radom.

Prepravite dijagram toka iz gornjeg primjera tako da ako upisano vrijeme **t** nije veće od nula da se na grani **NE** ispisuje poruka 'Vrijeme mora biti veće od nula' i tek tada odlazi na kraj. Pojedine znakove ili nizove znakova kao što je ova poruka u dijagramu toka pišemo unutar navodnika koji ovisno o programskom jeziku koji ćemo poslije koristiti mogu biti jednostruki ili dvostruki.

Idemo riješiti još jedan zadatak. Prisjetimo se zadatka s opsegom i površinom pravokutnika iz prethodnih vježbi koji ćemo još malo proširiti. Preko tipkovnice upisuju se stranice pravokutnika **a** i **b**. Treba izračunati opseg i površinu pravokutnika te ispisati poruku o tome da li je upisani lik pravokutnik ili kvadrat.



Nakon što smo upisali stranice a i b, izračunali smo opseg O i površinu P pravokutnika.

Dobivene rezultate možemo odmah ispisati.

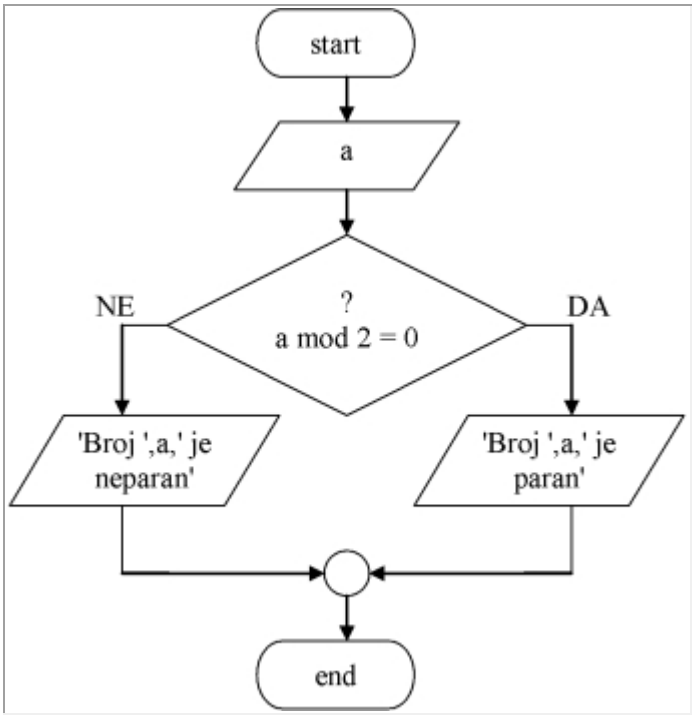
Kako ćemo utvrditi da li možda upisane stranice pripadaju kvadratu? Ako su stranice a i b jednake tada je to kvadrat.

Ispisujemo poruke i odlazimo na kraj.

Kraj.

Još malo grananja

Treba upisati cijeli broj **a** i ispisati poruku da li je upisani broj paran ili neparan.

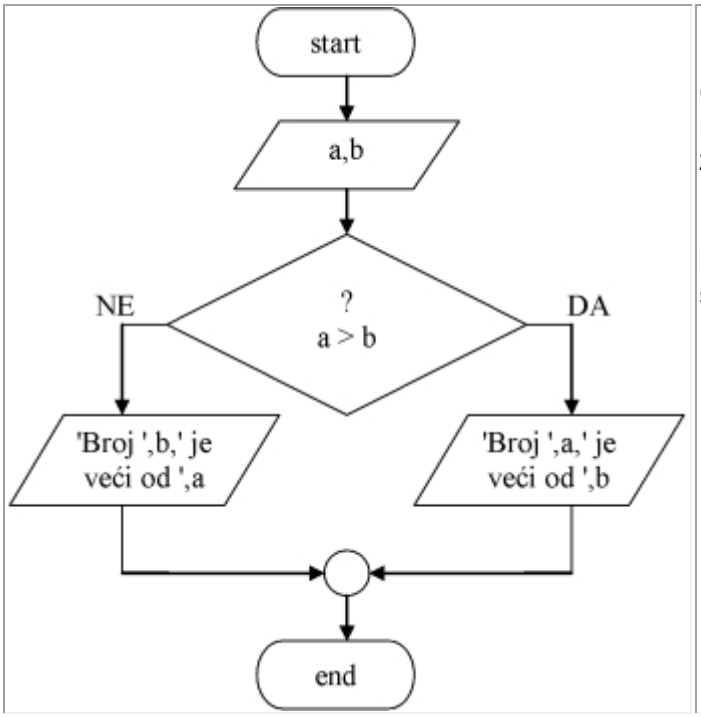


Upisali smo broj koji je smješten u varijablu **a**.

Kako saznati da li je **a** paran? Dijeliti ćemo ga s **2** koristeći operator **mod** koji kao rezultat daje ostatak cjelobrojnog dijeljenja. Ako je ostatak 0 tada je broj paran.

Obratite pažnju na konstrukciju ispisa. Pogledati ćemo lijevu granu koja se izvršava ako je odgovor na logički uvjet **NE**. Imamo dio poruke u navodnicima **'Broj '** pa zatim odvojeno zarezom varijablu **a** i nakon toga opet iza zareza drugi dio poruke u navodnicima **'je neparan.'** Ako je upisan npr. broj 9 tada će naredba za ispis **'Broj ',a,' je neparan'** rezultirati porukom koja izgleda kao rečenica u komadu: **Broj 9 je neparan**.

Treba upisati dva broja **a** i **b** i ispisati poruku o tome koji je veći.



Ovaj algoritam ima identične simbole i njihov raspored kao i algoritam u prethodnom zadatku. Razlikuje se pak u sadržaju.

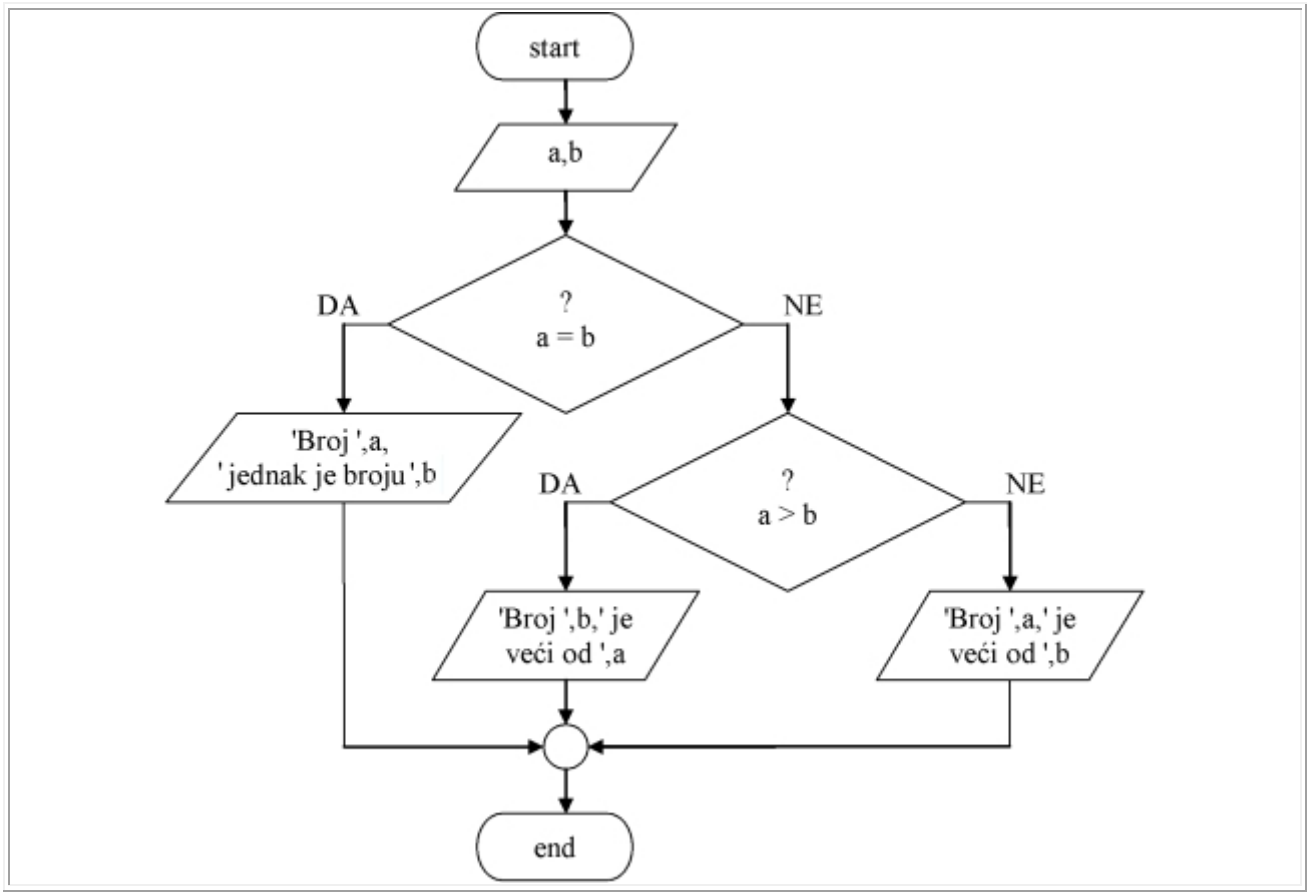
Koristeći relacijski operator **>** usporedili smo sadržaje varijabli **a** i **b**.

Stvar je zapravo vrlo jednostavna. Ali pokušajte sada zamisliti da netko upiše dva ista broja, npr. 7 i 7. U uvjetu ćemo imati pitanje da li je 7 veće od 7. Odgovor je **NE**. Izvršiti će se lijeva grana i kao rezultat dobiti ćemo poruku:

Broj 7 je veći od broja 7.

Ova poruka baš i nema logike i dokaz je da naše računalo od milja zvano idiot velike brzine i ne razmišlja baš puno. Zato ljudi koji rade na razvoju informacijskih sustava moraju predvidjeti čim više situacija, pa i onih najekstremnijih koje bi se pri radu programa mogle dogoditi. Kažu da dobra kontrola podataka pri unosu sprječava čak 90% grešaka koje bi se naknadno u radu programa mogle dogoditi.

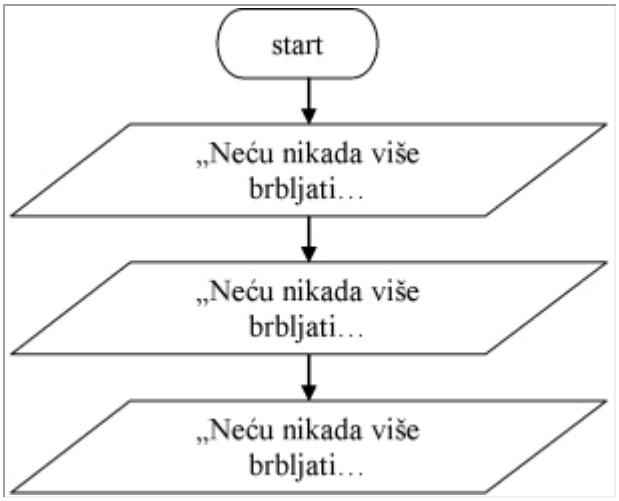
Zato ćemo mi rješenje našeg zadatka malo promijeniti:



Petlje

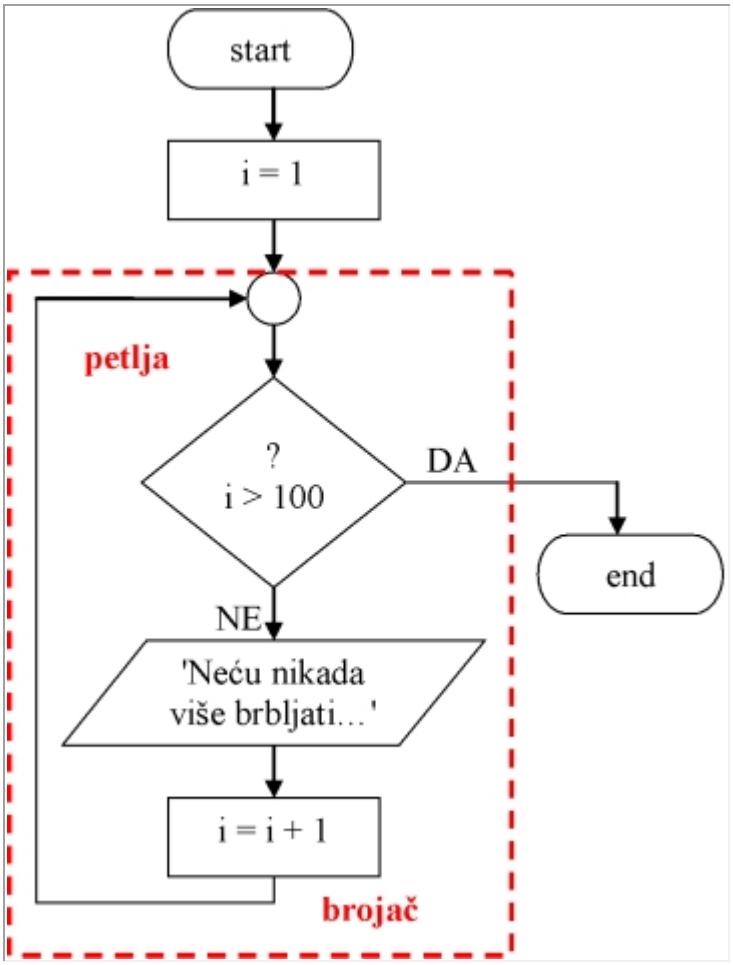
Naslov ove lekcije govori da sve ovo što smo do sada radili nije dovoljno zapetljano, pa ćemo stvari još malo dodatno zakomplicirati.

Nekada su nastavnici imali običaj nestašne učenike kažnjavati tako da su im zadali 100 puta napisati u bilježnicu: "Neću nikada više brbljati pod satom hrvatskog jezika!". Zamislite da je nama netko dao takvu kaznu. Mi ćemo se probati izvući tako da to računalo odradi umjesto nas. Treba mu samo zadati da 100 puta ispiše jednu te istu poruku. Ništa jednostavnije. Evo rješenja:



Hm, već nakon treće upute računalu da ispiše poruku sam odustao jer mi je dosadilo. Morate priznati da ovo rješenje baš i nije nešto s čime bi se mogli pohvaliti.

Pokušajmo to odraditi na drugačiji način:

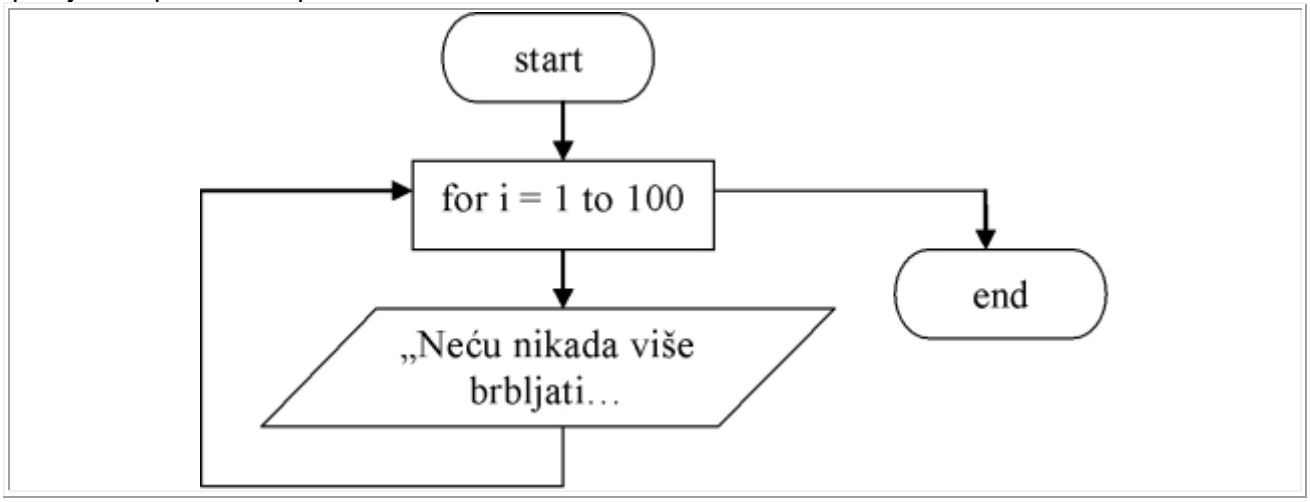


Varijablu **i** postavili smo na početnu vrijednost 1.

Nakon toga slijedi grananje u kojem postavljamo pitanje da li je **i** veći od 100. Naravno da nije jer smo ga upravo postavili na 1. Nastavljamo dakle granom **NE** i ispisujemo prvi puta našu poruku. Nakon toga varijablu **i** povećamo za 1. To smo već rekli da čitamo **i postaje i + 1**. Iza toga vraćamo se na grananje i ponovo pitamo da li je **i** veći od 100. Još uvijek nije jer je tek tren prije toga postao 2. Ponovno ispisujemo poruku i povećavamo **i** za 1 pa je on tako postao 3. I tako do 100 puta.

Nakon što se i 100-ti puta izvrše naredba grananja i naredbe ispod grananja **i** će postati 101. Tada je odgovor na pitanje u grananju **da li je i veći od 100** konačno **DA** i tada bi program trebao završiti s radom. Dio programa, niz istih naredbi koje se ponavljaju dok je neki uvjet zadovoljen ili dok ne postane zadovoljen, naziva se **petlja**. U našem se primjeru ispis poruke i povećanje varijable **i** za jedan ponavljaju dok **i** nije postao veći od 100. Varijabla **i** je u našem primjeru **brojač** jer svakim prolaskom kroz petlju **i postaje i više 1**.

Neki zlobnici tvrde da napredak možemo zahvaliti samo ljudskoj lijenosti. Naime, ljudi će svašta izmisliti samo da ne moraju raditi. Tako je netko izmislio još jednostavnije rješenje za naš primjer s ispisom 100 poruka.



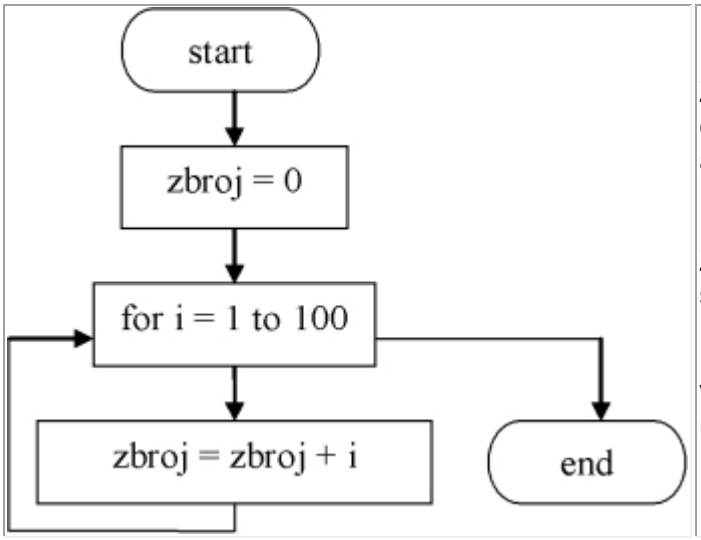
Pravokutnik ispod simbola za početak je naredba **for** koja je sama po sebi petlja. Brojač **i** kao i uvjet već su sadržani u naredbi pa ne moramo mi voditi brigu oko odbrojavanja. Taj red u algoritmu čitali bismo kao: "**Za i koji ide od 1 do 100 učini slijedeće:**".

Broj ponavljanja u petlji za rješenje određenih problema se zna unaprijed, prije početka ponavljanja, a za neke se ne zna unaprijed, već ovisi o izvršavanju niza naredbi koje se ponavljaju.

Ove druge petlje ćemo ostaviti za drugi razred kada ćemo se malo detaljnije baviti programiranjem. Za sada ćemo se samo baviti petljama u kojima unaprijed znamo broj ponavljanja, dakle petljom **for**.

Još malo petljanja

Koliko vremena vam treba da zbrojite sve brojeve od 1 do 100? Prava sitnica.



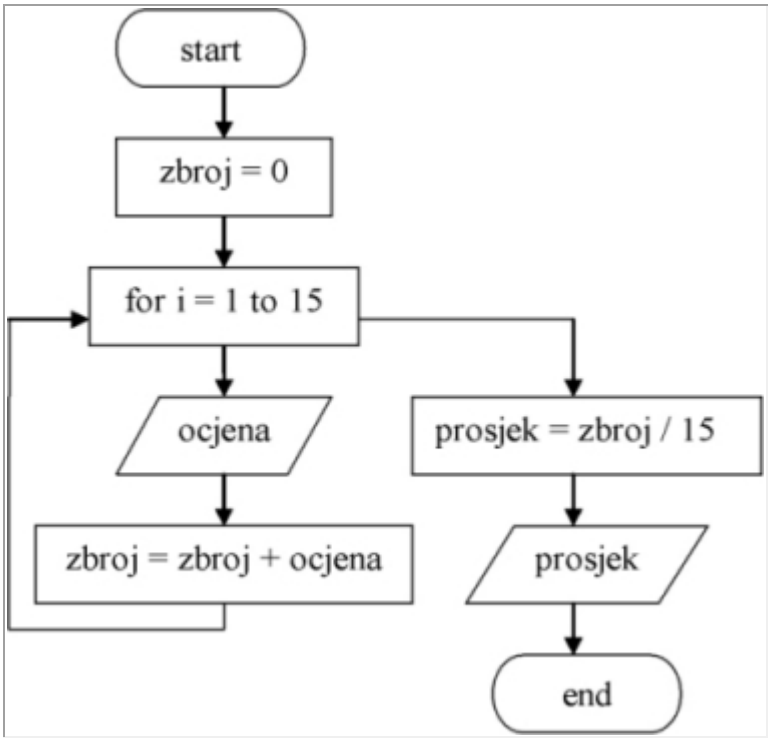
Za naš zbroj svih brojeva do 100 koristiti ćemo varijablu **zbroj** koju na početku algoritma moramo postaviti na vrijednost 0.

Za **i** koji ide od 1 do 100 treba učiniti slijedeće:

Varijabli **zbroj** dodati vrijednost varijable **i** koja je u našem primjeru brojač.

Svakim prolaskom kroz petlju **i** se povećava za 1 i njegova vrijednost pridodaje sadržaju varijable **zbroj**. U prvom prolazu kroz petlju **zbroj** je 0, a **i** je 1, pa prema tome **zbroj** postaje 1. U drugom prolazu **zbroj** je 1, a **i** je 2. **Zbroj** postaje 1+2, dakle sada je 3. I tako 100 puta.

Idemo još malo zapetljati nešto što znamo računati još od petog osnovne, a to je prosjek ocjena. Prosjek je zbroj svih ocjena podijeljen s brojem predmeta.



Varijablu **zbroj** koristiti ćemo za zbroj svih ocjena i moramo ju na početku postaviti na vrijednost 0.

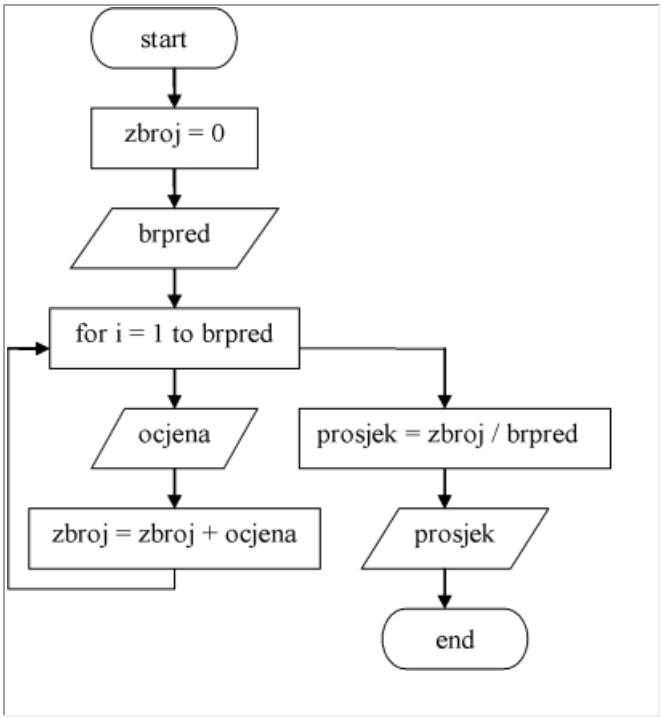
Neka u prvom razredu imamo 15 predmeta. Varijablu **i** postavili smo na početnu vrijednost 1.

U prvom prolazu kroz petlju upisujemo prvu ocjenu iz imenika, a to je hrvatski jezik. Ocjena se upisuje u varijablu **ocjena**. Sadržaj varijable **ocjena** se dodaje u varijablu **zbroj**.

U drugom prolazu upisujemo ocjenu iz drugog predmeta, u trećem iz trećeg i tako 15 puta. Po izlazu iz petlje izračunavamo prosjek i ispisujemo ga.

Ovo je bilo prejednostavno da bi bilo stvarno. U stvarnom životu desiti će se da je netko n pr. oslobođen nastave tjelesne i zdravstvene kulture pa ima 14 predmeta. Netko možda ima 16 predmeta jer je izabrao i fakultativnu nastavu. Isto tako u drugom, trećem ili četvrtom razredu sigurno nećete imati 15 predmeta. Već smo napomenuli da se programi pišu čim općenitije tako da

bi se mogli primjenjivati za rješavanje čim više sličnih problema. Zato ćemo naš algoritam prepraviti tako što ćemo umjesto 15 uvesti varijablu **brpred** koja će predstavljati naš broj predmeta.



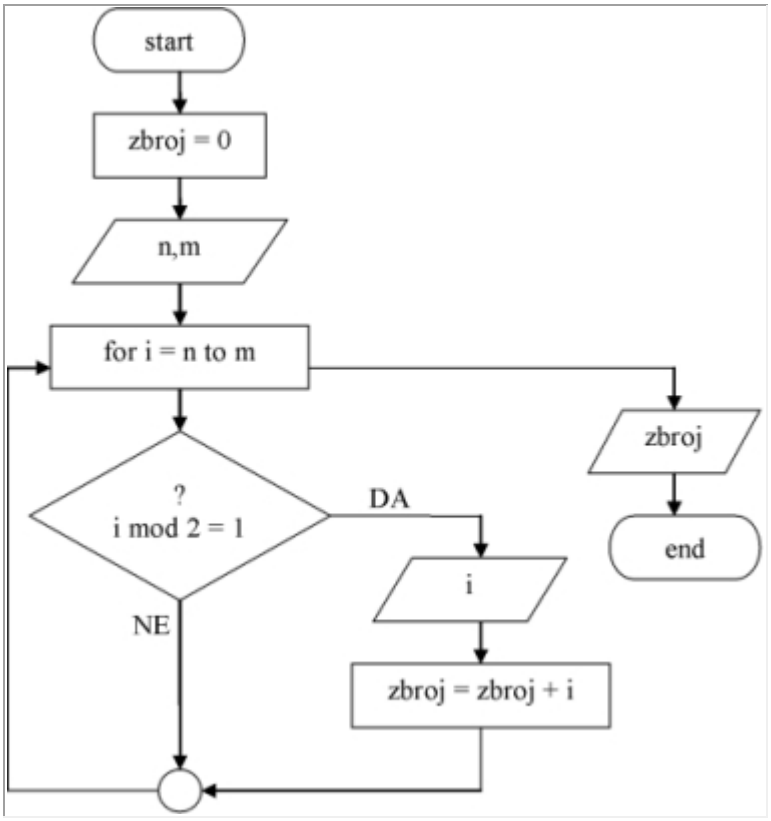
Nakon što smo varijablu **zbroj** postavili na početnu vrijednost 0 treba preko tipkovnice unijeti broj predmeta koji se smješta u varijablu **brpred**.

Brojač **i** u petlji sada broji od 1 do **brpred** pa je primjenjivo za sve učenike i razrede s različitim brojem predmeta.

Da bismo dobili **prosjek** treba po izlasku iz petlje **zbroj** ocjena podijeliti s brojem predmeta **brpred**.

Za vježbu: Pokušajte po uzoru na prosjek ocjena za jednog učenika napraviti dijagram toka za izračun prosječne ocjene razreda.

Malo ćemo kombinirati petlje s grananjima. Sjetimo se zadatka u kojem smo morali zbrojiti sve brojeve od 1 do 100. Promijeniti ćemo taj zadatak tako da sada treba ispisati sve neparne brojeve od n do m i na kraju ispisati njihov zbroj.



Nakon postavljanja varijable **zbroj** na početnu vrijednost 0 treba u upisati početni broj **n** i završni broj **m** od kojeg do kojeg želimo ispisati i zbrojiti neparne brojeve.

Brojač u petlji **for** sada broji od **n** do **m**

Da li je **i** neparan broj znati ćemo ako ga podijelimo s 2 koristeći operator **mod** i dobijemo rezultat 1.

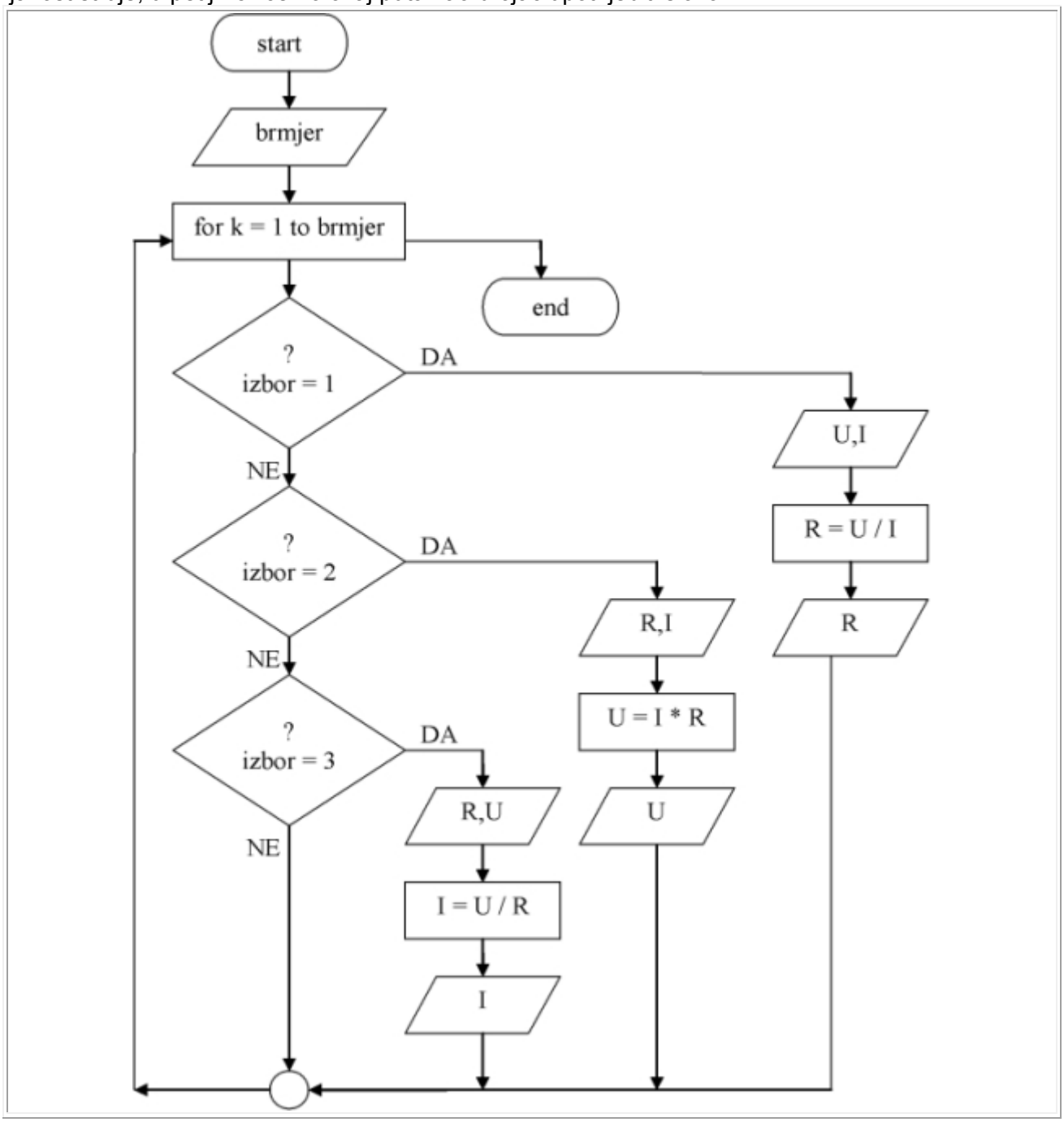
Ako ostatak je 1 tada ispisujemo **i** i dodajemo njegovu vrijednost varijabli **zbroj**.

Što će se dogoditi ako netko upiše za n veći broj nego za m?
Prepravite gornji algoritam tako da izračunava i zbroj neparnih i zbroj parnih brojeva od n do m.

Učenici su na satu fizike naučili Ohmov zakon po kojem otpor u istosmjernom strujnom krugu možete izračunati tako da podijelite napon sa jakošću struje:

R = U / I

gdje je otpor R, napon U i jakost struje I. Naučeno su odmah provjerili mjereći veličine u strujnom krugu, ako su mjerili napon i jakost mogli su po gornjoj formuli izračunati otpor, ako su mjerili otpor i napon mogli su izračunati jakost struje **I = U / R**, a ako su mjerili otpor i jakost struje mogli su izračunati napon **U = I * R**. Treba napraviti algoritam po kome je moguće za određeni broj mjerenja na temelju dvije izmjerene veličine izračunati treću. Za broj mjerenja ćemo koristiti varijablu **brmjer**. Uvesti ćemo još jednu varijablu koju ćemo zvati **izbor** i dogovoriti ćemo se da u nju upišemo 1 ako želimo računati otpor, 2 ako želimo računati napon i 3 ako želimo računati jakost struje. Ako upisani broj ne bude 1, 2 ili 3 ne treba poduzimati ništa. Pošto slovo **I** koristimo kao varijablu za jakost struje, u petlji **for** ćemo ovaj puta kao brojač upotrijebiti slovo **k**.



Ovaj primjer može se daleko elegantnije riješiti višestrukim grananjem, ali to ostavljamo za slijedeći razred.

Zaključak

Izrada algoritama zahtjeva puno vježbe. Algoritmi se ne mogu naštrebati. Potrebno je riješiti puno primjera, od jednostavnijih do sve složenijih.

Dijagram toka crtamo običnom olovkom s brisalom pri ruci. Stvari se rješavaju u hodu, mijenjaju, popravljaju i prepravljaju. Uočavaju se stvari koje se mogu pojednostavniti. Prednost dijagrama toka je što je pregledan, lako se test primjerima može "pješice" provjeriti. Zanimljivo je i to da više ljudi može napraviti na isti zadatak više različitih algoritama i da su svi dobri. Složeni algoritmi mogu se rastaviti na manje, tako rastavljeni riješiti do detalja i onda prikazati kao rješenje u blokovima. Problemi se mogu lako analizirati, lako se uočavaju sličnosti i razlike između više rješenja i odabire najbolje.

Možda nakon ovih vježbi vidite dijagrame toka i tamo gdje ih do sada niste uočavali - različiti shematski prikazi nekih postupaka, hodogrami nekih aktivnosti...

I na kraju što je bio cilj naše izrade algoritama - pisanje programa. Kada je jednom algoritam gotov pisanje programa u nekom programskom jeziku je prepisivanje simbola iz dijagrama toka u naredbe programskog jezika. Obično svaki simbol u dijagramu toka predstavlja jednu naredbu u programu. Dijagram toka bi trebao biti dio svake dobre programske dokumentacije.

Učenicima je crtanje dijagrama toka ponekad nerazumljivo, a sve što je nerazumljivo je i dosadno. Oni bi najradije sjeli za računalo i odmah pisali program. Kod jednostavnih problema to i možemo napraviti. Ali kod složenijih... Jednom sam sreo svog bivšeg učenika koji se u svom poslu intenzivno bavi programiranjem. Žalio mi se da nekad nekoliko sati provede za računalom pokušavajući riješiti neki problem, ali ne ide i gotovo. Pitao sam ga što tada učini. Odgovorio je: "Pa uzmem olovku i papir i nacrtam algoritam..."

Preuzeto sa: http://public.carnet.hr/~zorkovac/informatika/algoritmi/Algoritmi.html

Dodatak: Programi za izradu dijagrama toka programa na računalu
MS Word, može poslužiti
MS Visio, to mu je i namjena

Diagram Designer, jednostavan programčić, omogućuje spremanje uradtka u .jpg, .gif, .pd i drugim formatima. Program je free.