Cvetana Krstev

Algoritmi i programiranje

Jezički prevodioci

Jezički prevodioci, kao primer opšteg softvera, obezbeđuju da se tekst programa zapisanog na nekom programskom jeziku prevede na mašinski jezik (tj. na takve instrukcije koje određeni procesor može da izvrši). Programom se zapisuju algoritmi koji predstavljaju skup dobro definisanih pravila ili instrukcija za rešavanje nekog problema (npr. za obavljanje izračunavanja), u konačnom broju koraka. Izražavanje algoritma u formalnoj notaciji je jedan od glavnih zadataka programa. Ali, algoritam zapisan u nekom programskom jeziku, kao što je paskal, C ili Python ne može se u tom obliku i izvršiti: neophodno je prevesti ga na skup instrukcija koje su definisane za određeni procesor. Te instrukcije su oblika „dopremi iz memorije u akumulator podatak sa određene adrese" ili „dodaj na sadržaj akumulatora sadržaj sa određene memorijske lokacije", a izražene su kao sekvencije bitova — u obliku nepodesnom za ljudsko kodiranje i korišćenje.

Odnos između jednog algoritma i njegovog zapisa na mašinskom jeziku prolazi sledeće dve faze: u prvoj, programer izražava algoritam u određenom programskom jeziku, a u drugoj ga prevodi na mašinski jezik, koristeći odgovarajući jezički prevodilac.

Programski jezici mogu biti različitog nivoa. Tradicionalno se razlikuju četiri osnovna nivoa programskih jezika.

- *mašinski jezik*, jedini koji je „razumljiv" računaru, je jezik najnižeg nivoa. On je uvek specifičan određenom tipu procesora. Programiranje na mašinskom jeziku podrazumeva ispisivanje instrukcija računara kao sekvencija 0 i 1, u redosledu njihovog izvršavanja.
- *asemblerski jezici*, su vrlo bliski mašinskim jezicima, i takođe su specifični za određeni tip procesora. Umesto da se ispisuju sekvencije 0 i 1 uvode se skraćenice za sve mašinske instrukcije (kao, na primer, `ADD` za instrukciju sabiranja ili `MOV` (od engl. *move*) za premeštanje podataka između unutrašnje memorije računara i registara u centralnom procesoru). Osim toga, programer može da dodeli simbolička imena memorijskim adresama na kojima se nalaze podaci sa kojima program radi kao i adresama na kojima se nalaze instrukcije samog programa. Asemblerski jezici u principu zahtevaju jedan red asemblerskog koda za jednu mašinsku instrukciju. Specijalni jezički procesori, koji se nazivaju *asembleri*, prevode program zapisan na asemblerskom jeziku u mašinski jezik.

- *razvijeni programski jezici* (ili viši programski jezici ili jezici 3. generacije) su jezici poput jezika paskal, C ili Fortran. U njima kontrolne strukture i strukture podataka odražavaju potrebe algoritma, a ne zahteve hardvera. Programski jezici ove vrste su često standardizovani na međunarodnom nivou, što znači da se programi napisani na tako standardizovanim jezicima mogu izvršavati, praktično bez izmene, na računarima sa različitim procesorima i pod različitim operativnim sistemima. Specijalni jezički procesori, koji se nazivaju *kompilatori* i *interpretatori* prevode program zapisan na nekom razvijenom jeziku u mašinski jezik. Ovi jezički procesori često generišu mnogo mašinskih instrukcija za jedan iskaz u programu na razvijenom programskom jeziku.
- *jezici četvrte generacije*, su neproceduralni jezici jer se pomoću njih ne izražava algoritam, odnosno, procedura za rešavanje nekog problema. Programer treba samo da izrazi svoje zahteve koje jezički prevodilac pretvara u odgovarajuću proceduru. Ovi jezici, na primer, omogućavaju prikaz i pretraživanje informacija, na primer, u bazama podataka ili na Internetu. Primer ovakvog jezika je SQL – Standard Query Language.

Algoritmi

Algoritam predstavlja specifikaciju koraka koje treba slediti da bi se neki problem rešio u konačnom broju koraka. Algoritmi su veoma bliski računarskim programima, i ponekad se s njima poistovećuju. Algoritmom se, međutim, mogu opisati mnogi postupci koji se nikada neće izvršavati putem računara. Tipičan primer su kuvarski recepti. Svaki kuvarski recept počinje opisom ulaznih sastojaka, zatim sledi precizan opis upotrebe tih sastojaka kao i uslova za završetak operacije kuvanja. Na primer, recept za pripremu voćnog kolača (koji se na francuskom zove *clafoutis* - klafuti) glasi ovako:

> Potrebna su 4 sveža jajeta, malo soli, 1 paket vanilin-šećera, pola čaše kristal-šećera, 4 kašike brašna (najbolje oštrog), pola čaše mleka, pola čaše kisele pavlake, 30gr putera i sezonsko ili smrznuto voće, najbolje crne trešnje, višnje, kajsije, kruške a može i mešano. Vatrostalnu posudu koja ide u rernu posutu kristal-šećerom i u nju naređati voće da prekrije celo dno posude. Mikserom umutiti cela jaja sa malo soli. Dodati vanilin-šećer i kristal-šećer i dobro umutiti. Polako dodavati brašno i dalje mutiti. Zatim dodati mleko i umutiti. Dodati pavlaku i umutiti. Na kraju dodati otopljen puter (npr. u mikrotalasnoj, bez kuvanja). Tečnu masu preliti preko voća i staviti u rernu da se peče na

temperaturi od 180-200 stepeni. Kolač je gotov kada dobije intenzivnu boju, stegne se i počne intenzivno da miriše.

Mnogi od algoritama koji se danas koriste u računarskim programima nastali su mnogo pre nastanka računara. Poznat je *Euklidov algoritam* za određivanje najvećeg zajedničkog delioca (NZD) dva cela broja. Najveći zajednički delilac dva cela broja je najveći ceo broj koji bez ostatka deli oba broja. Na primer, 21 je NZD brojeva 252 i 105 jer je to najveći ceo broj koji deli bez ostatka oba broja: $252 = 21 \cdot 12$, $105 = 21 \cdot 5$. (Kako znamo da je 21 najveći takav broj? Činioci brojeva 252 i 105, osim broja 21, su 12 i 5 koji su uzajamno prosti brojevi.) Najjednostavniji postupak za rešenje ovog problema je da se počne sa manjim od dva data broja te da se unazad ispituju svi brojevi sve dok se ne dođe do onog koji deli oba polazna broja. *Euklid*, grčki matematičar koji je živeo u III veku pre nove ere, postavio je drugi algoritam koji je opisao u VII i X knjizi svog čuvenog dela *Elementi*.[1] Taj algoritam glasi:

> Neka treba naći najveći zajednički delilac celih brojeva m i n. (To je najveći broj koji deli bez ostatka i jedan i drugi broj.) Ako je $m>n$, treba podeliti m sa n i neka je r ostatak pri deljenju. Ako je $r=0$ onda je rešenje n; inače, treba isti algoritam ponoviti sa n i r.

Ovaj algoritam zasniva se na činjenici da ako neki broj deli bez ostatka brojeve m i n, $m>n$, onda taj isti broj deli bez ostatka i brojeve n i r, pri čemu je r ostatak pri deljenju m sa n, što se može i formalno pokazati.

> Ako je k broj koji deli i broj m i n, onda se ta dva broja mogu zapisati u obliku $m = m_1 \cdot k$ i $n = n_1 \cdot k$. Deljenje brojeva m i n možemo da predstavimo u obliku $m = Q \cdot n + r$, gde je Q celobrojni deo količnika a r ostatak pri deljenju. Kada zamenimo m i n sa gornjim izrazima dobijamo
> $m_1 \cdot k = Q \cdot n_1 \cdot k + r$, odnosno
> $r = k \cdot (m_1 - Q \cdot n_1)$, što znači da broj k deli ostatak r.

U slučaju brojeva $m = 252$ i $n = 105$ i broja $k = 21$, to znači da ako podelimo m sa n, možemo rezultat predstaviti u obliku $m = Q \cdot n + r$, tj. $252 = 2 \cdot 105 + 42$, a broj 21 takođe deli bez ostatka broj 42 jer je $42 = 2 \cdot 21$.

[1] Videti animaciju Euklidovof algoritma na srpskoj i engleskoj Vikipediji: https://sr.wikipedia.org/sr-el/Еуклидов_алгоритам i https://en.wikipedia.org/wiki/Euclidean_algorithm

Ovaj algoritam je mnogo efikasniji od "naivnog" algoritma jer u najvećem broju slučajeva u mnogo manjem broju koraka dolazi do rešenja. Posmatrajmo slučaj brojeva m=25 i n=15. Sledeći naivan algoritam postupili bismo na sledeći način:

Nzd	15	14	13	12	11	10	9	8	7	6	5
ost. m/nzd	10	11	12	1	3	5	7	1	4	1	0
ost. n/nzd	0	1	2	3	4	5	6	7	1	3	0

Do rešenja *nzd*=5 dolazi se u 11 koraka, pri čemu se u svakom koraku obavljaju dva deljenja. Ako se primeni Euklidov algoritam za isti par brojeva m=25 i n=15 do rešenja se dolazi u samo 3 koraka.

m	25	*m*←*n*	15	10
n	15	*n*←*r*	10	5
r	10		5	0

Poznato je dosta takvih algoritama nastalih daleko pre nastanka računara. Pomenimo, na primer, algoritam poznat pod nazivom *Eratostenovo sito* koji precizira efikasan postupak za pronalaženje prostih brojeva. *Eratosten* je bio grčki matematičar, geograf i astronom iz III veka pre nove ere. Poznat je i algoritam koji se pripisuje napuljskom astronomu *Aloysius Lilius* i nemačkom jezuitskom matematičaru *Christopher Clavious* a koji potiče s kraja XVI veka. Njihov postupak većina zapadnih crkava koristi za određivanje Uskrsa za svaku godinu posle 1582.[2]

Za precizan zapis algoritma često se koristi neka vrsta pseudo-formalnog jezika koji obezbeđuje utvrđivanje korektnosti algoritma i njegovo prevođenje u konkretan programski jezik. Pod korektnošću algoritma se podrazumeva da algoritam za precizno zadate ulazne vrednosti u konačnom broju koraka daje precizno specifikovan izlazni rezultat. Algoritam za spravljanje voćnog kolača bi se mogao zapisati koristeći ovaj pseudo-jezik na sledeći način:

```
Pripremi materijal
Izaberi voće
Poređaj voće u posudu
Umuti jaja i so
```

[2] Uskrs bi trebalo da bude „prva nedelja posle prvog punog meseca koji dolazi posle 21. marta". Ima mnogih indicija da je jedina važnija upotreba aritmetike u srednjem veku u Evropi bila računanje datuma Uskrsa. Prema tome, po gregorijanskom kalendaru Uskrs može da bude najranije 22. marta, a najkasnije 25. aprila, odnosno po julijanskom kalendaru najranije 3. aprila, a najkasnije 8. maja (što je bilo 1983. godine).

Dodaj vanilin-šećer i umuti
Dodaj kristal-šećer i umuti
Dodaj brašno i umuti
Dodaj mleko i umuti
Dodaj pavlaku i umuti
Dodaj puter i umuti
Ponavljaj
    peci u rerni na 180-200º
sve dok ne požuti **i** ne stegne se **i** ne zamiriše

Koristeći sličan pseudo-formalni specifikacioni jezik, Euklidov algoritam bi se mogao zapisati na sledeći način:

m i *n* su celi brojevi
r je ostatak pri deljenju *m* sa *n*
Sve dok je *r* različito od 0 **ponavljaj odavde**
    m dobija vrednost od *n*
    n dobija vrednost od *r*
    r je ostatak pri deljenju *m* sa *n*
do ovde
Najveći zajednički delilac *nzd* je *n*

Zadaci

Zadatak 1. Izračunati najmanji zajednički sadržalac (NZS) dva broja. To je najmanji broj koji se može podeliti bez ostatka sa oba data broja.

Prvi broj je *b1*
Drugi broj je *b2*
m dobija vrednost *b1*
n dobija vrednost *b2*
r je ostatak pri deljenju *m* sa *n*
sve dok je *r* različito od 0 **ponavljaj odavde**
    m dobija vrednost *n*
    n dobija vrednost *r*
    r je ostatak pri deljenju *m* sa *n*
do ovde
NZD je broj *n*
NZS je *b1***b2*/*NZD*

Primer: b_1 je 34, b_2 je 51. Najjednostavnije rešenje je da se uzme da je ZS = b_1* b_2 = 1734, ali da li je to najmanji broj koji sadrži i jedan i drugi broj?

m	51	*m*←*n*	34
n	34	*n*←*r*	**17**
r	17		0

NZS = 1734/17 = 102 (102=34*3; 102=51*2)

Podsećanje: Zašto su nam potrebni *NZD* i *NZS*? Recimo da želimo da saberemo razlomke 1/34 i 1/51. Moramo ih dovesti na zajednički sadržalac, najbolje na najmanji: 1/34=3/102, 1/51=2/102, dakle 1/34 + 1/51 = 3/102 + 2/102 = 5/102. Ukoliko ne bismo koristili *NZS* onda bi sabiranje izgledalo: 51/1734 + 34/1734 = 85 / 1734. Sada bi bilo potrebno da se „skrati“ ovaj razlomak, a za to je potrebno odrediti *NZD* brojeva 85 i 1734, npr. Euklidovim algoritmom:

m	1734	*m*←*n*	85	34
n	85	*n*←*r*	34	**17**
r	34		17	0

Sada je 85 = 17 · 5, a 1734 = 17 · 102 pa dobijamo konačan rezultat 1/34 + 1/51 = **5/102**.

Zadatak 2. Da li je *e* prost broj?

f dobija vrednost 2
Ako je *NZD*(*e*,*f*) različito od 1 **onda** *e* nije prost broj **i kraj**
Inače uradi odavde
    f je broj 3
    Sve dok je *f* manje ili jednako √*e* **i**
    NZD(*e*,*f*) je jednako 1 **ponavljaj odavde**
        f je broj za 2 veći
    do ovde
do ovde
Ako je *NZD*(*e*,*f*) različito od 1
    onda *e* nije prost broj
    inače *e* je prost broj
kraj

(a) f je potencijalni faktor broja e. Ako se u toku rada algoritma ustanovi da je *NZD* brojeva e i f različit od 1, onda to znači da broj *NZD* deli broj e bez ostatka pa prema tome broj e nije prost broj.

(b) Zašto se u petlji broj f (potencijalni faktor) uvećava za 2? Na početku se ispituje deljivost broja e s brojem 2 i ako se ustanovi da broj 2 ne deli broj e, onda to znači da e nije paran broj pa deljivost sa drugim parnim brojevima više ne treba ispitivati.

(c) Zašto se ispituju samo faktori f koji su manji ili jednaki kvadratnom korenu iz e? Ako broj e ima neki faktor f_1 veći od $\sqrt{e}$ onda je $e = f_1 * f_2$ pri čemu je f_2 sigurno faktor manji od $\sqrt{e}$. Uzmimo za primer 144 = 12 * 12. Faktori broja 12 su 2, 2, 3, pa su faktori broja 144 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48 i 72, a broj 144 se može zapisati kao proizvod dva faktora – dok je prvi faktor manji od kvadratnog korena broja drugi faktor je veći, kada prvi faktor postane veći od kvadratnog korena broja drugi faktor je manji:

144 = 2 * 72
144 = 3 * 48
144 = 4 * 36
144 = 6 * 24
144 = 8 * 18
144 = 9 * 16
144 = 12 * 12
144 = 16 * 9
144 = 18 * 8
...

Šta nam to znači? Ako nijedan broj manji od $\sqrt{e}$ nije faktor broja e, a to znamo jer smo sve te moguće faktore ispitali, nije sigurno ni neki veći broj.

Primer: e je 41

f	2	3	5	7
		$3 \leq 6.4$	$5 \leq 6.4$	$7 \geq 6.4$
$NZD(41,f)$	1	1	1	

f=2 $m \leftarrow 41, n \leftarrow 2, r \leftarrow 1 \neq 0$
$m \leftarrow 2, n \leftarrow 1, r = 0$, kraj, $NZD \leftarrow 1$
f=3 $m \leftarrow 41, n \leftarrow 3, r \leftarrow 2 \neq 0$
$m \leftarrow 3, n \leftarrow 2, r \leftarrow 1 \neq 0$
$m \leftarrow 2, n \leftarrow 1, r = 0$, kraj, $NZD \leftarrow 1$
f=5 $m \leftarrow 41, n \leftarrow 5, r \leftarrow 1 \neq 0$
$m \leftarrow 5, n \leftarrow 1, r = 0$, kraj, $NZD \leftarrow 1$

$NZD(41,f) = 1 \rightarrow 41$ je prost broj

Program

Da bi se neki algoritam mogao izvršavati na računaru potrebno je da on bude zapisan na nekom programskom jeziku koji određeni računar „razume“. Programski jezik je notacija za precizan opis računarskih programa ili algoritama. Programski jezici su veštački jezici, u kojima su sintaksa i semantika strogo definisani. Dakle, svrsishodni su ali ne dozvoljavaju slobodu izražavanja koja je svojstvena prirodnom jeziku.

Da bi programiranje bilo moguće, potrebno je, takođe, predstaviti podatke s kojima program radi na računaru prihvatljiv način. Ovaj drugi zahtev odbacuje algoritam za pripremu voćnog kolača kao računarski neprimenljiv jer se njegovi „podaci“ — jaja, so, šećer, itd. — ne mogu računarski predstaviti. Euklidov algoritam je, pak, vrlo pogodan za zapisivanje u obliku programa i izvršavanje na računaru.

Algoritmi se za potrebe izvršavanja na računaru zapisuju u obliku programa na nekom programskom jeziku. Da bi zapisivanje algoritama na programskom jeziku bilo efikasno programski jezik treba da obezbeđuje rad sa *osnovnim tipovima podataka* kao i *strukture podataka* i *kontrolne strukture*. Pod osnovnim tipovima podataka podrazumevaju se celobrojni, realni, karakterski i logički (ili bulovski) tip. O načinu predstavljanja ovih osnovnih tipova, osim logičkog, bilo je već reči. Logički tip podatka je jednostavan za predstavljanje jer je potrebno predstaviti samo dve vrednosti: tačno i netačno. Stoga se logički tip obično predstavlja u jednoj memorijskoj lokaciji tako da recimo binarna vrednost 0 označava netačno a binarna vrednost 1 tačno (ili svaka vrednost koja nije 0 tačno).

U višim programskim jezicima, kakvi su C, paskal i Fortran, a takav je i VBA, programer može da dodeli ime nekim memorijskim lokacijama i istovremeno da deklariše kakav tip podatka će u njima biti zabeležen. Ovde je bitno da se uoče dve stvari: programer dodeljuje ime memorijskim lokacijama a pri tome uopšte ne mora znati, i najčešće to ni ne zna, kojim memorijskim lokacijama, to jest kojim adresama, su ta imena pridružena. Osim toga, u imenovanoj memorijskoj lokaciji se uvek nalazi sekvencija 0 i 1, i tek joj programska deklaracija daje određenu interpretaciju. Ovako imenovane memorijske lokacije u koje se smeštaju vrednosti određenog tipa koje se mogu i menjati u toku izvršavanja programa nazivaju se *promenljive*. U programskom jeziku Python promenljive se ne deklarišu – u pitanju je netipizirani programski jezik. To ne znači da promenljive nemaju tip – on se određuje na osnovu vrednosti koja je promenljivoj dodeljena.

U donjoj tabeli prikazani su osnovni tipovi podataka, kako se deklarišu promenljive određenog tipa, a kako se zapisuju konstantne vrednosti tog tipa. Svi

primeri u ovom priručniku odnosiće se na programski jezik VBA pomoću koga se mogu zapisivati makroi u okviru aplikacija *Word*, *Excel* i *PowerPoint* i na programski jezik Python.

Tipovi podataka

Tip	VBA	Konstante
celobrojni (2B)	`Dim x As Integer`	`3 -5 -32768 32767`
celobrojni (4B)	`Dim y As Long`	`3 -5 0 -2147483648`
realni (jednostruka tačnost)	`Dim z As Single`	`50.3  3.14E-4`
realni (dvostruka tačnost)	`Dim u As Double`	`2.0E+79 -1.427E+12`
logički	`Dim ind As Boolean`	`True False`
tekstuelni	`Dim s As String`	`"Pera"  "++//--=="`

Kada se kaže da programski jezik treba da obezbedi rad sa osnovnim tipovima podataka to, pre svega, znači da su u programskom jeziku definisane osnovne *operacije* nad određenim tipom podataka, kao i osnovne *relacije*. Na primer, ako su `A` i `B` celobrojne promenljive, onda se sa `A+B` označava operacija sabiranja celobrojnih vrednosti koje su smeštene u memorijskim lokacijama koje su imenovane sa `A` i `B`. Rezultat ove operacije je zbir tih celobrojnih vrednosti. Nad celobrojnim tipom podatka obično se definišu unarne operacije promene znaka, te binarne operacije sabiranja, oduzimanja, množenja i celobrojnog deljenja i računanja po modulu, odnosno određivanja ostatka pri deljenju i operacija promene znaka. Nad realnim tipom podatka definišu se unarne operacije promene znaka, te binarne operacije sabiranja, oduzimanja, množenja i deljenja. Nad logičkim tipom podatka definišu se binarne operacije konjunkcije i disjunkcije i unarna operacija negacije, dok se nad tekstualnim tipom definiše operacija konkatenacije ili dopisivanja.

U većini programskih jezika nad brojčanim, celobrojnim i realnim tipom kao i nad karakterskim tipom definisane su binarne relacije poređenja: jednako, nejednako, manje, veće, manje ili jednako i veće ili jednako. Svaka relacija takođe ima rezultat, a to je logička vrednost `True` ili `False` u zavisnosti od toga da li su dve vrednosti u relaciji ili nisu. Na primer, vrednost izraza `5<3` je `false` jer celobrojna vrednost 5 nije manja od celobrojne vrednosti 3, to jest 5 nije u relaciji „manje od“ sa 3. Karakterske vrednosti se takođe mogu porediti, s tim što treba imati u vidu da se u stvari porede njihove binarne reprezentacije, pa vrednost relacionog izraza zavisi od pozicije karaktera koji se porede u kolacionoj sekvenciji. Na primer, vrednost izraza `"a"<"A"` je `false` ako računar koristi ASCII kôd (Latin-1, Unicode), a `true` ako računar koristi EBCDIC kôd.

Operatori

Aritmetički operatori (celi i realni brojevi)	**VBA i Python**
- (promena znaka)	`-A`
+, -, *	`A+B, A-B, A*B`
\ // (količnik pri celobrojnom deljenju)	`7 \ 3 = 2 ; 7 // 3 (rezultat je 2)`
Mod % (ostatak pri celobrojnom deljenju)	`7 Mod 3 = 1 ; 7 % 3 (rezultat je 1)`
/ (realno deljenje)	`7 / 3 (rezultat je 2.33333...)`
^ ** (stepenovanje)	`2 ^ 4 = 16 ; 2**4 (rezultat je 16)`
Relacijski operatori (operatori poređenja)	**VBA i Python**
= ==	`A = B ; A == B`
<> **!=** (nejednakost)	`A <> B ; A != B`
<	`A < B`
<=	`A <= B`
>	`A > B`
>=	`A >= B`
Logički operatori	**VBA i Python**
And **and** (konjunkcija)	`(godina Mod 4 = 0) And (godina Mod 100 <> 0)` `(godina % 4 == 0) and (godina % 100 != 0)`

Or **or** (disjunkcija)	`(godina Mod 400 = 0) _`
	`Or _`
	`((godina Mod 4 = 0) _`
	`  And _`
	`(godina Mod 100 <> 0))`
	(Ceo uslov se može zapisati u jednoj liniji)
	`(god % 400 == 0) or ((god % 4 == 0) and (god % 100 != 0))`

Not **not** (negacija)	`Not (godina mod 4 = 0)  ' godina mod 4 <> 0` `not (god % 4 == 0)  # god % 4 != 0`
Tekstuelni operatori	**VBA i Python**
& **+** (konkatenacija)	`"Pera" & " i " & "Mika" ' "Pera i Mika"` `"Pera" + " i " + "Mika" # "Pera i Mika"`

Operacije i relacije definisane u nekom jeziku mogu se kombinovati u složene izraze. Mogu se koristiti i zagrade, na sličan način kako je to uobičajeno u matematici da bi se prevazišli prioriteti operacija. Operacije sabiranja i oduzimanja su nižeg prioriteta od operacija množenja i deljenja pa stoga, na primer, izrazi `3*2+5` i `3*(2+5)` nemaju istu vrednost (vrednost prvog je 11 a drugog 21). Među logičkim operacijama, kao i među relacijama su prioriteti izvršavanja takođe definisani, a prioriteti koji važe u programskim jezicima VBA i Python su dati u donjoj tabeli.

Prioritet operatora

Prioritet operatora (od višeg ka nižem)	
VBA	**Python**
^ (stepenovanje)	** (stepenovanje)
- (promena znaka)	- (promena znaka)
* /	* / % (moduo) // (celobrojno deljenje)
\ (celobrojno deljenje)	
Mod	
+ -	+ -
& (konkatenacija)	
= < <= > >= <>	== < <= > >= !=
And Or Not	not
	and
	or

Aritmetički izrazi

Aritmetički izrazi se postupno definišu na sledeći način:

1. Celobrojne i realne konstante su aritmetički izrazi.
2. Celobrojne i realne promenljive (tipa Integer, Long, Single i Double) su aritmetički izrazi.
3. Pozivi celobrojnih i realnih funkcija su aritmetički izrazi.
4. Ako su E_1 i E_2 aritmetički izrazi, tada su aritmetički izrazi i:

VBA	**Python**	
$E_1 + E_2$	$E_1 + E_2$	(zbir)
$E_1 - E_2$	$E_1 - E_2$	(razlika)
$E_1 * E_2$	$E_1 * E_2$	(proizvod)
$E_1 \setminus E_2$	$E_1 // E_2$	(količnik pri celobrojnom deljenju)
E_1 Mod E_2	$E_1 \% E_2$	(ostatak pri celobrojnom deljenju)
E_1 / E_2	E_1 / E_2	(količnik pri deljenju realnih brojeva)
$E_1 \wedge E_2$	$E_1 ** E_2$	(stepenovanje)

(E_1)	(E_1)	(izraz u zagradama)
$+E_1$	$+E_1$	(potvrda znaka)
$-E_1$	$-E_1$	(promena znaka)

Primer. Kako će se u programskom jeziku VBA Python izračunati vrednost izraza:

`7*3-6\2+4`	`7*3-6//2+4`

Vrednost izraza se računa s leva u desno, s tim što se operacije višeg prioriteta izračunavaju pre onih koje su nižeg prioriteta:

VBA	Python
`21-6\2+4`	`21-6//2+4`
`21-3+4`	`21-3+4`
`18+4`	`18+4`
`22`	`22`

Iskaz dodele

Vrednost ovih složenih izraza može se dodeliti kao vrednost nekoj promenljivoj. To je smisao osnovnog programskog iskaza koji je prisutan u većini programskih jezika, a to je *iskaz dodele*. Promenljiva kojoj se dodeljuje vrednost može i sama učestvovati u izrazu čija se vrednost dodeljuje. Smisao iskaza dodele je da se izračunata vrednost smešta na memorijsku lokaciju kojoj je pridružena promenljiva kojoj se dodeljuje vrednost. S leve strane operatora dodele se, dakle, nikad ne može pojaviti izraz, već samo ime promenljive koje u tom slučaju ima značenje memorijske adrese na koju izračunatu vrednost izraza s desne strane operatora dodele treba upisati. Naredba dodele u programskom jeziku VBA ima opšti oblik:

`promenljiva = izraz.`

Primer. Neka je dat sledeći kratak VBA program.

```
Dim n As Integer
n = 2
n = n + n * n Mod 3 + n \ (n - 1) * (n * n)
```

_____

Prikazati postupno kako se izračunava izraz s desne strane iskaza dodele i šta je konačna vrednost promenljive `n`.

```
2 + 2 * 2 Mod 3 + 2 \ (2 - 1) * (2 * 2)
```

Gledano s leva nadesno, najvišeg prioriteta je množenje

```
2 + 4 Mod 3 + 2 \ (2 - 1) * (2 * 2)
```

Gledano s leva nadesno, najvišeg prioriteta je množenje, ali prvo treba da se izračunaju izrazi u zagradama

```
2 + 4 Mod 3 + 2 \ 1*(2 * 2)
2 + 4 Mod 3 + 2 \ 1 * 4
2 + 4 Mod 3 + 2 \ 4
```

Gledano s leva nadesno, najvišeg prioriteta je celobrojno deljenje
```
2 + 4 Mod 3 + 0
```
Gledano s leva nadesno, najvišeg prioriteta je ostatak pri celobrojnom deljenjenju
```
2 + 1 + 0
```
Gledano s leva nadesno, najvišeg prioriteta je sabiranje
```
3 + 0
```
Gledano s leva nadesno, najvišeg prioriteta je sabiranje
```
3
```
Naredba dodele
```
n ← 3
```

U Python-u
```
n = 2
n = n + n * n % 3 + n // (n – 1) * (n * n)
```

_____

Prikazati postupno kako se izračunava izraz s desne strane iskaza dodele i šta je konačna vrednost promenljive `n`.
```
2 + 2 * 2 % 3 + 2 // (2 – 1) * (2 * 2)
```
Gledano s leva nadesno, najvišeg prioriteta je množenje
```
2 + 4 % 3 + 2 // (2 – 1) * (2 * 2)
```
Gledano s leva nadesno, najvišeg prioriteta je moduo (ostatak pri deljenju)
```
2 + 1 + 2 // (2 – 1) * (2 * 2)
```
Gledano s leva nadesno, najvećeg prioriteta je celobrojno deljenje, ali prvo treba da se izračuna izraz u zagradama
```
2 + 1 + 2 // 1*(2 * 2)
2 + 1 + 2 * (2 * 2)
```
Gledano s leva nadesno, najvišeg prioriteta je množenje, ali prvo treba da se izračuna izraz u zagradama
```
2 + 1 + 2 * 4
2 + 1 + 8
```
Gledano s leva nadesno, najvišeg prioriteta je sabiranje
```
3 + 8
```
Gledano s leva nadesno, najvišeg prioriteta je sabiranje
```
11
```
Naredba dodele
```
n ← 11
```

Primer. Razmotriti sledeći program za razmenu vrednosti dve promenljive i utvrditi zašto sledeće rešenje nije dobro.

Pogrešno rešenje	
VBA	**Python**
`Dim a As Integer, b As Integer`	`a = 5`

a = 5
b = 7
a = b ' a ← 7
b = a ' b ← 7
	b = 7
	a = b # a ← 7
	b = a # b ← 7

Ispravno rešenje bi glasilo:

Ispravno rešenje	
VBA	**Python**

Dim a, As Integer, b As Integer, p As Integer
a = 5
b = 7
p = b ' p ← 7
b = a ' b ← 5
a = p ' a ← 7
	a = 5
	b = 7
	p = b # p ← 7
	b = a # b ← 5
	a = p # a ← 7

Primeri (U narednim primerima za rešavanje primera koriste se samo aritmetičke operacije)

1. Zadati četvorocifreni prirodni broj transformisati tako da se iz njega izbaci druga cifra (po težini, tj. druga s desne strane):

VBA	x = 2731 x1 = x \ 10 ' x1 ← 2731 \ 10 => x1 ← 273 c1 = x Mod 10 ' c1 ← 2731 Mod 10 => c1 ← 1 x2 = x1 \ 10 ' x2 ← 273 \ 10 => x2 ← 27 y = x2 * 10 + c1 ' y ← 27 * 10 + 1 => y ← 271

Python	x = 2731
	x1 = x // 10 #x1=273 **prve 3 cifre**
	c1 = x % 10 #c1=1 **poslednja cifra**
	x2 = x1 // 10 #x2=27 **prve 2 cifre**
	y = x2*10 + c1 #y=271

Kraće rešenje bi bilo:

VBA	Dim x As Integer, y As Integer x = 2731 y = (x\100)*10 + x Mod 10 ' y ← 27 * 10 + 1 => y ← 271
Python	x = 2731 y = (x//100)*10 + x % 10

2. Zadati četvorocifreni prirodni broj transformisati tako da cifre na poziciji 2 i 3 zamene mesta:

VBA

```
Dim x As Integer, x1 As Integer, x2 As Integer, _
    x3 As Integer, c1 As Integer, c2 As Integer, _
    c3 As Integer, y As Integer
X = 2731
x1 = x \ 10           '    x1 ← 2731 \ 10 => x1 ← 273
c1 = x Mod 10    '    c1 ← 2731 Mod 10 => c1 ← 1
x2 = x1 \ 10          '    x2 ← 273 \ 10 => x2 ← 27
c2 = x1 Mod 10   '    c2 ← 273 Mod 10 => c2 ← 3
x3 = x2 \ 10          '    x3 ← 27 \ 10 => x3 ← 2
c3 = x2 Mod 10   '    c3 ← 27 Mod 10 => c3 ← 7
y = x3 * 1000 + c2 * 100 + c3 * 10 + c1
'    y ← 2 * 1000 + 3 * 100 + 7 * 10 + 1 => y ← 2371
```

Python

```
x = 2731
x1 = x // 10          #x1 ← 273   prve 3 cifre
c1 = x % 10           #c1 ← 1     poslednja cifra
x2 = x1 // 10         #x2 ← 27    prve 2 cifre
c2 = x1 % 10          #c2 ← 3     pretposlednja cifra
x3 = x2 // 10         #x3 ← 2     prva cifra
c3 = x2 % 10          #c3 ← 7     druga cifra
y = x3 * 1000 + c2 * 100 + c3 * 10 + c1
                      #y ← 2*1000+3*100+7*10+1
```

Kraće rešenje bi bilo:

VBA

```
Dim x As Integer, y As Integer
x = 2731
y = (x \ 1000)*1000 + ((x \ 10) Mod 10)*100 + ((x \ 100) Mod
10)*10 + (x Mod 10)
```

Python

```
x = 2731
y = (x // 1000)*1000 + ((x // 10) % 10)*100 +
((x // 100) % 10)*10 + (x % 10)
```

3. Neka je X celobrojna promenljiva čija je vrednost pozitivan broj od tačno 5 cifara. Napisati u Pajtonu (VBA) deo programa koji određuje broj Y kod koga je izbačena treća cifra po redu (cifra stotina). Na primer, za broj X=12468 program treba da odredi Y=1268. Ovaj deo programa može da koristi pomoćne promenljive (ne treba ispisivati deklaracije promenljivih).

VBA

```
Dim x As Integer, x1 As Integer, c1 As Integer, y As Integer
x1 = x \ 1000    ' x1 = 12468\1000 = 12
c1 = x Mod 100   ' c1 = 12468 Mod 100 = 68
y = x1*100 + c1  ' 12*100+68=1268
```

Python	x = 12468
	x1 = x // 1000 #x1 ← 12 **prve 2 cifre**
	c1 = x % 100 #c1 ← 68 **poslednje 2 cifre**
	y = x1*100 + c1 #12*100+68=1268

Kraće rešenje bi bilo:

VBA	Dim x As Integer, y As Integer x = 12468 y = (x \ 1000) * 100 + x Mod 100 '(12468 \ 1000)*100 + 12468 Mod 100=1268
Python	x = 12468 y = (x // 1000) * 100 + x % 100 #(12468 // 1000)*100+12468%100=1268

Relacijski izrazi

Neka su E_1 i E_2 aritmetički izrazi (ili neka su E_1 i E_2 karakterski izrazi). Tada su relacijski izrazi i:

VBA	**Python**	
$E_1 = E_2$	$E_1 == E_2$	(upoređivanje da li su vrednosti dva aritmetička izraza jednake)
$E_1 <> E_2$	$E_1 != E_2$	(upoređivanje da li su vrednosti dva aritmetička izraza različite)
$E_1 < E_2$	$E_1 < E_2$	(upoređivanje da li je vrednost prvog aritmetičkog izraza manja od vrednosti drugog)
$E_1 <= E_2$	$E_1 <= E_2$	(upoređivanje da li je vrednost prvog aritmetičkog izraza manja ili jednaka vrednosti drugog)
$E_1 > E_2$	$E_1 > E_2$	(upoređivanje da li je vrednost prvog aritmetičkog izraza veća od vrednosti drugog)
$E_1 >= E_2$	$E_1 >= E_2$	(upoređivanje da li je vrednost prvog aritmetičkog izraza veća ili jednaka vrednosti drugog)

Primeri

1. Šta će biti vrednost bulovske promenljive posle izvršavanja sledećeg dela programa:

VBA	Dim B As Boolean B = (22 Mod 3) ^ 3 <= 4 ^ 2 \ 5
Python	b = (22 % 3) ** 3 <= 4 ** 2 // 5 Sve relacijski operatori su nižeg prioriteta od aritmetičkih operatora. Vrednost izraza E1 je: (22 % 3) ** 3 -> 1 ** 3 -> 1 Vrednost izraza E2 je: 4 ** 2 // 5 -> 16 // 5 -> 3

	Vrednost promenljive B je: `1 <= 3 -> true`

2. Šta će biti vrednost sledećih bulovskih promenljivih posle izvršavanja sledećeg dela programa:

```
Dim a As Boolean, b As Boolean, c As Boolean, _
    d As Boolean
a = "Vlada" < "vlada"
b = "vlada5" < "vladalac"
c = "vlada-5" < "vlada-V"
d = "ABC" < "AB_C"
```

Sve četiri promenljive će imati vrednost true jer se poređenje zasniva na ASCII kodu (Unicode-u).

Logički izrazi

1. Logičke konstante True i False su logički izrazi.
2. Logičke promenljive (tipa Boolean) su logički izrazi.
3. Relacijski izrazi su logički izrazi.
4. Pozivi logičkih funkcija su logički izrazi.
5. Ako su L_1 i L_2 logički izrazi, tada su logički izrazi i
   - L_1 And L_2 L_1 and L_2
   - L_1 Or L_2 L_1 or L_2
   - Not L_1 not L_1

Primer

Sastaviti logički izraz koji određuje da li je neka godina prestupna. Godina je prestupna ako je deljiva sa 4, ali godine deljive sa 100 nisu prestupne, s izuzetkom godina deljivih sa 400 koje jesu prestupne.

```
(g Mod 400 = 0) Or ((g Mod 4 = 0) And (g Mod 100 <>0))
(g % 400 == 0) or ((g % 4 == 0) and (g % 100 != 0))
```

Na primer, za godinu 2012. ovaj izraz se računa na sledeći način (na isti način u VBA i Python-u):

```
(2012 % 400 == 0) or ((2012 % 4 == 0) and (2012g % 100 != 0))
(12 == 0) or ((2012 % 4 == 0) and (2012 % 100 != 0))
False or ((2012 % 4 == 0) and (2012 % 100 != 0))
False or ((0 == 0) and (2012 % 100 != 0))
False or (True and (2012 % 100 != 0))
False or (True and (12 != 0))
False or (True and True)
False or True
True
```

Zadaci

1. Zapisati iskaz u jeziku VBA koji dodeljuje promenljivoj *dali* vrednost `True` ili `False` u zavisnosti od toga da li je vrednost promenljive Z jednaka zbiru promenljivih X i Y (ne treba ispisivati deklaracije promenljivih).

```
dali = (z == (x + y))
```

Za `z==5`, `x==2` i `y==1`, `dali` dobija vrednost `False`

Za `z==12`, `x==5` i `y==7`, `dali` dobija vrednost `True`

2. Neka je x celobrojna promenljiva čija je vrednost pozitivan trocifreni broj. Napisati deo programa koji određuje da li se broj završava sa dve iste cifre. Npr. za broj 722 bulovska promenljiva *dane* treba da dobije vrednost `True`, a za broj 721 bulovska promenljiva *dane* treba da dobije vrednost `False`. Ovaj deo programa može da koristi pomoćne promenljive (ne treba ispisivati deklaracije promenljivih).

VBA

```
x = 288
c1 = x Mod 10                ' c1 = 8
c2 = (x \ 10) Mod 10         ' c2 = 28 Mod 10 = 8
dane = (c1=c2)               ' dane = (8 = 8) = true
```

Python

```
x = 288
c1 = x % 10                  #c1 ← 8  prva cifra
c2 = (x // 10) % 10          #c2 ← 8  druga cifra
dane = (c1==c2)              #dane = (8 == 8) ← true
                             ili kraće
dane = ((x % 10) == ((x //10) % 10))
```

Konverzija tipova

Većina programskih jezika dozvoljava da binarnim i relacijskim operatorima budu povezane i vrednosti različitog tipa. Tako se, na primer, binarna operacija sabiranja može primeniti na jednu celobrojnu i jednu realnu vrednost. Međutim, da bi se operacija izvršila celobrojna vrednost se pretvara u realnu, što znači da se binarna vrednost iz oblika potpunog komplementa pretvara u oblik pokretnog zareza. O ovom pretvaranju se stara jezički procesor i programer o tome ne mora da vodi računa. Rezultat operacije je realna vrednost. Ovo važi za sve binarne operacije i za sve relacije. Kombinovanje brojčanih i karakterskih tipova aritmetičkim operatorima nije dozvoljeno, osim ako karakterska niska nije takva da se može tumačiti kao brojčana konstanta. Kombinovanje brojčanih i karakterskih tipova karakterskim operatorom je dozvoljeno i rezultat je uvek karakterska niska, pri

čemu se brojčana vrednost pretvara u odgovarajući zapis. Prilikom naredbe dodele, VBA pokušava da konvertuje tip izraza sa desne strane u tip promenljive sa leve strane znaka jednakosti. U Python-u se niske i broje ne mogu kombinovati u jednom izrazu već se jedan od operanada mora eksplicitno konvertovati u drugi tip korišćenjem odgovarajuće ugrađene funkcije u zavisnosti od toga šta želimo da obavimo.

Primer. Kako će se u programskom jeziku izračunati vrednost sledećeg izraza vodeći računa da se vrednost izraza računa s leva u desno, s tim što se operacije višeg prioriteta izračunavaju pre onih koje su nižeg prioriteta.

VBA	`1.2 +19 Mod 6\2+4` ‘ najvišeg prioriteta je celobrojno deljenje
	`1.2 +19 Mod 3 +4` ‘ najvišeg prioriteta je ostatak pri deljenju
	`1.2 + 1 +4` ‘ pre sabiranja treba izvršiti konverziju
	`1.2 + 1.0 + 4`
	`2.2 + 4` ‘ pre sabiranja treba izvršiti konverziju
	`2.2 + 4.0`
	`6.2`

Python	`1.2 + 19 % 6 // 2 + 4 #najvećeg prioriteta %`
	`1.2 + 1 // 2 +4 #najvećeg prioriteta //`
	`1.2 + 0 + 4 #pre sabiranja konverzija`
	`1.2 + 0.0 + 4`
	`1.2 + 4 #pre sabiranja konverzija`
	`1.2 + 4.0`
	`5.2`

Pravila o konverziji tipova i razne primere daje sledeći programski odlomak:

VBA

```
Dim ix As Integer, ry As Double, s As String   'operandi
Dim iz As Integer, iv As Integer, rw As Double,
    s2 As String                               'rezultati
Dim poruka As String
ix = 5 'inicijalizacija
ry = 4.3 'inicijalizacija
s = "123" 'inicijalizacija
iz = ix * ry ' 5 * 4.3 => 5.0 * 4.3 => 21.5 => 21
rw = ix * ry ' 5 * 4.3 => 5.0 * 4.3 => 21.5
s2 = s & ix  ' "123" & 5 => "123" & "5" => "1235"
iv = ix + s  ' 5 + "123" => 5 + 123 => 128
MsgBox ix + s ' 5 + "123" => 5 + 123 => 128 => "128"
s = "a123"
iv = ix + s ' Greška!! "a123" ne može da se konvertuje u broj
MsgBox ix + s ' Greška!! "a123" ne može da se konvertuje u broj
s = "123.4"
rw = ix + s ' 5 + "123.4" => 5 + 123.4 => 5.0 + 123.4 = 128.4
iv = ix + s ' 5 + "123.4"  => 5+123.4 => 5.0 + 123.4 = 128.4 => 128
MsgBox ix + s ' 5+"123.4" => 5+123.4 => 5.0+123.4=128.4 => "128.4"
```

Python

```
ix = 5         #inicijalizacija
ry = 4.3       #inicijalizacija
s = '123'      #inicijalizacija
rz = ix * ry       # 5 * 4.3 => 5.0 * 4.3 => 21.5
iw = int(ix * ry)  # 5 * 4.3 => 5.0 * 4.3 => 21.5 => 21
ss = s + ix        #Greška
ss = s + str(ix)   # "123" + str(5) => "123" + "5" => "1235"
iy = int(s) + ix   # int("123") + 5 => 123 + 5 => 128
rw = 123.4
sss = "32.4"
rq = rw + float(s)        # 123.4 + float("123") => 123.4 + 123.0 => 246.4
iq = ix + int(float(sss)) # 5+int(float(32.4)) => 5+int(32.4) => 5 + 32 => 37
```

Navedeni primeri pokazuju da se podrazumevaju pravila poput:

- `Integer * Double => Double`;
- `String & Integer => String`;
- prilikom naredbe dodele, VBA pokušava da konvertuje tip izraza sa desne strane u tip promenljive sa leve strane znaka jednakosti;
- prilikom poziva funkcije (npr. `MsgBox ix`) VBA pokušava da konvertuje tip stvarnog argumenta (`ix As Integer`) u tip parametra naveden u definiciji te funkcije (u slučaju funkcije `MsgBox`, prvi parametar mora da bude tipa `String`).

Za slučaj `String + Integer` vidimo da rezultat zavisi od vrednosti tipa `String`:

- ako je vrednost tipa String validan tekstuelni zapis celog broja ili broja u pokretnom zarezu, moguće je izvršiti konverziju iz tipa `String` u odgovarajući celobrojni ili realni tip; u tom slučaju se izraz svodi na `Integer + Integer => Integer` ili `Double + Integer => Double`
- ako vrednost tipa String nije validan tekstuelni zapis ni celog broja ni broja u pokretnom zarezu, prijavljuje se greška.

Ovakva podrazumevana pravila se još nazivaju pravila implicitne konverzije tipova.

Zadaci

1. Koju vrednost dobija promenljiva *dali* u ovom delu programa:

VBA

```
Dim i As Integer, x As Single, s As String, _
    j As Integer, dali As Boolean
   i = 12
   x = 5.3
   s = "-7"
   j = i + x + s
   dali = (j = 10)
```

	Kako se izračunava iskaz:
	`j = i + x + s`
	`j = 12 + 5.3 + "-7" 'prvo sabiranje, potrebna konverzija`
	`j = 12.0 + 5.3 + "-7"`
	`j = 17.3 + "-7"  'drugo sabiranje, potrebna konverzija`
	`j = 17.3 + (-7.0)  'potrebna konverzija`
	`j = 10.3  'dodela, potrebna konverzija`
	`'j <- 10`
	`'dali <- true`

Python	`i = 12`
	`x = 5.3`
	`s = "-7"`
	`j = i + x + float(s)`
	`dali = (j == 10)`
	`#j = 12 + 5.3 + float("-7") :potrebna konverzija`
	`#j = 12.0 + 5.3 +float("-7")`
	`#j = 17.3 + float("-7")  :potrebna konverzija`
	`#j = 17.3 + (-7.0)`
	`#j = 10.3`
	`#dali = (10.3 == 10)  :potrebna konverzija`
	`#dali = (10.3 == 10.0)  :dali <- False`