Univerza v Mariboru

Fakulteta za energetiko

RAČUNALNIŠTVO ZA INŽENIRJE ENERGETIKE

skripta

GORAZD HREN

ALENKA HREN

2013

Gorazd Hren, Alenka Hren

Računalništvo za inženirje energetike

Naslov publikacije: Računalništvo za inženirje energetike

Vrsta publikacije: Skripta

Dostopni na spletni strani: https://estudij.uni-mb.si/course/view.php?id=9150

Avtorja: dr. Gorazd Hren

dr. Alenka Hren

Recenzent: prof.dr. Anton Jezernik

Izdajatelj: Založniška dejavnost Fakultete za energetiko

Leto objave: maj 2013

Naslov odgovornega avtorja: gorazd.hren@um.si

Predgovor

Publikacija je namenjena študentom tehnike v začetnih letnikih fakultete, ko se morajo spomniti, da računalnik ni samo za igranje igrc in da tudi mobilniki in podobne naprave zahtevajo poznavanje osnov računalništva in programiranja. Uporabniki naprav, ki poznajo osnove delovanja so bistveno bolj učinkoviti pri uporabi naprav.

Gradivo je namenjeno študentom, ki nimajo predznanja o računalništvu, programiranju in končni aritmetiki. Učbeniki s področja računalništva se z razvojem računalništva zelo spreminjajo. V osemdesetih letih je učenje računalništva temeljilo na učenju všjenivojskih programskih jezikov, kot so fortran ali pascal. Pozneje z izrednim porastom uporabe računalnika so postale pomembnejše aplikacije na področjih ekonomije, medicine in izobraževanja. Z razvojem osebnih računalnikov je učenje računalništva temeljilo na uporabi orodij za pisanje tekstov, obvladovanje preglednic, predstavitev in elektronske pošte. V zadnjem času je poudarek na uporabi spleta, spletnih aplikacij in tehnologij kot so HTML, XML in Java. Računalništvo se vrača k osnovam, saj današnji študentje že prihajajo z znanjem o uporabi osebnih računalnikov, elektronske pošte in uporabniških programov. Mnogi so že sestavljali spletne strani in imajo osebne spletne strani. Študenti in kasneje inženirji so zahtevnejši uporabniki računalnika, saj je računalnik in progamje nepogrešljivo orodje za inženirsko delo, kar hkrati pomeni, da morajo poznati osnove računalništva. V tem gradivu je kar nekaj računalništva, prevladuje pa uporaba računalnika.

Avtorja dokazujeva sinergijo delovanja pri pripravi študijskega gradiva in upava, da bodo bralci prepoznali namen in trud avtorjev.

Maribor, 2013

avtorja

Kazalo vsebine

1 UVOD

Ta tekst je povabilo v eno od najmlajših in atraktivnih znanstvenih disciplin – računalništvo (*angl. Computer Science*).

Povprečni študent intuitivno pozna dovolj dobre opise znanstvenih disciplin, čeprav jih niso študirali. Biologija je študij o živih organizmih in kemija opisuje strukturi in sestavi materialov. Redko je poznavanje kaj je računalništvo. Pravzaprav je znanih kar nekaj napačnih interpretacij.

Računalništvo je proučevanje računalnikov.

Ta trditev je napačna ali pomanjkljiva. Že delo znanstvenikov med letoma 1920 in 1940 je podalo teoretične osnove računalnika (logika in matematika), čeprav računalnika takrat še sploh ni bilo. Razliko me računalnikom in računalništvom sta dobro opisala Fellows in Parberry[1] v reviji Computing Research News:

»Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes, or chemistry is about beakers and test tubes. Science is not about tools. It is about how we use them and what we find out when we do.«

Računalništvo je študij kako zapisati in uporabljati računalniške programe.

Ker je programiranje računalnikov zelo vezano na računalnik, se velikokrat pojma računalništva in pisanja programov enačita. Programiranje je v računalništvu izredno pomembno, vendar je enako kot računalnik le orodje za preverjanje novih metod in rešitev problemov, kot je na primer predstavitev informacije. Tipičen primer je ena od najpogostejših aplikacij v računalništvu, posebej prisotna v velikih sistemih, iskanje imena v velikih seznamih. Če razvijemo bolj učinkovito in hitrejšo metodo preiskovanja jo razvijemo teoretično in nato s programiranjem prenesmo v računalnik, kjer lahko izmerimo učinkovitost, je to pravo računalništvo.

O računalništvu je zapisanih veliko definicij, vendar je naslednja najbogatejša: **osnovni koncept računalnika je algoritem**.

Razvoj algoritma za reševanje vrste problemov sestavljajo štiri osnovne operacije:

Proučevanje obnašanja algoritma, da preverimo, če je točen in učinkovit (formalne in matematične lastnosti)
Razvoj računalniških sistemov, ki so sposobni algoritem izvajati (strojna oprema).
Razvoj programskih jezikov in predstavitev algoritmov v prigramskih jezikih, ki so sposobni izvajati algoritme s strojno opremo.
Identificirati probleme in izvesti točne in učinkovite programske rešitve za reševanje problemov (aplikacije).

Računalništvo je veda o algoritmih, vključno z njihovimi formalnimi in matetmatičnimi lastnostmi, njihovo realizacijo s strojno opremo in programskimi jeziki ter uporabo.

Inženir mora biti pri svojem delu učinkovit in računalnik je v veliko pomoč, če ga zna uporabljati:

Inženir mora znati uporabljati osebni računalnik, delovno postajo in računalniška omrežja.
Zavedati se mora prednosti in slabosti sodobnih informacijskih tehnologij.
Znati mora uporabljati standardne programe.
Sposoben se mora naučiti novih znanj in spoznanj na področju računalništva in programiranja ter jih vključevati v svoje strokovno delo.
Obvladati mora tehniko iskanja in procesiranja informacij.

[1] Fellows, M. R., and Parberry, I. “Getting Children Excited About Computer Science,” *Computing Research News*, vol. 5, no. 1 (January 1993).

Znati mora s pomočjo računalnikov reševati tudi zahtevnejše naloge.

Zradi vseh naštetih razlogov inženir spada med zahtevnejše uporabnike računalniško podprtih tehnologij.

1.1 Algoritem in diagram poteka

al.go.rithm je postopek za reševanje matematičnega problema v končnem številu korakov, ki mnogokrat vključuje veliko število ponavljanja operacij. Na kratko: korak po korak definirana metoda za reševanje problema.

Za sestavo algoritma potrebujemo tri kategorije operacij:

Zaporedne operacije: ena operacija definira natančno en enostavno izvedljiv korak. Ko je korak končan se izvajanje nadaljuje na naslednjem koraku. Velikokrat so to prireditveni izrazi: dodaj žlico sladkorja v skodelico s kavo; predpiši spremenljivki x vrednost 1.

Pogojne operacije: so instrukcije algoritma vezane na »vprašanje – odgovor«. Postavi se vprašanje izvajanje se nadaljuje na podlagi odgovora (če je zahtevek za izplačilo manjši ali enak stanju na računu izplačaj zahtevom, drugače obvesti lastnika, da je prekoračil stanje; če je x ni enak 0, izvedi račun 1/x, drugače izpiši napako – deljenje z ničlo).

Iterativne operacije: to so zankaste instrukcije algoritmov. Povedo, da se izvajanje na nadaljuje na naslednjem koraku, temveč se vrne in ponovno izvaja blok instrukcij (ponavljaj z večanjem števca, dokler ne doseže predpisane vrednosti).

Algoritme uporabljamo ves čas, čeprav se tega ne zavedamo vedno: navodila za sestavljanje pohištva, otroških igrač, kuhanje po receptih, postopek vpisovanja na fakuleto in še bi lahko naštevali.

Dobro in logično programiranje je proces razvoja programa, ki se prične s planiranjem in organiziranjem dela. Vsak problem lahko rešimo z izvajanjem niza aktivnosti v določenem vrstnem redu. Izvajane aktivnosti in njihov vrstni red imenujemo algoritem. Izkušnje dokazujejo, da je najtežji del programiranja problema na računalniku prav razvijanje algoritma za reševanje. Ko je algoritem pripravljen in pravilen je kodiranje sorazmerno enostavno. Naslednji primer prikazuje pravilen in nepravilen vrstni red aktivnosti:

Odkleni avto, Sedi v avto, Vključi motor, **Spusti ročno zavoro** , Spelji

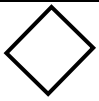
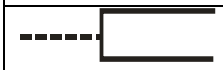
Odkleni avto, Sedi v avto, Vključi motor, Spelji, **Spusti ročno zavoro**

Algoritem je torej spisek navodil, ki v končnem številu korakov privede do rezultata, pri čemer je vsak korak dobro znana operacija. Algoritem večinoma predstavimo tudi v grafični obliki zaradi lažjega preverjanja toka izvajanja v programu. Algoritem dovolj natančno opisuje postopek pri reševanju problema, vhodne in izhodne podatke, ponavljanje operacij, ipd., ni pa odvisen od programskega jezika.

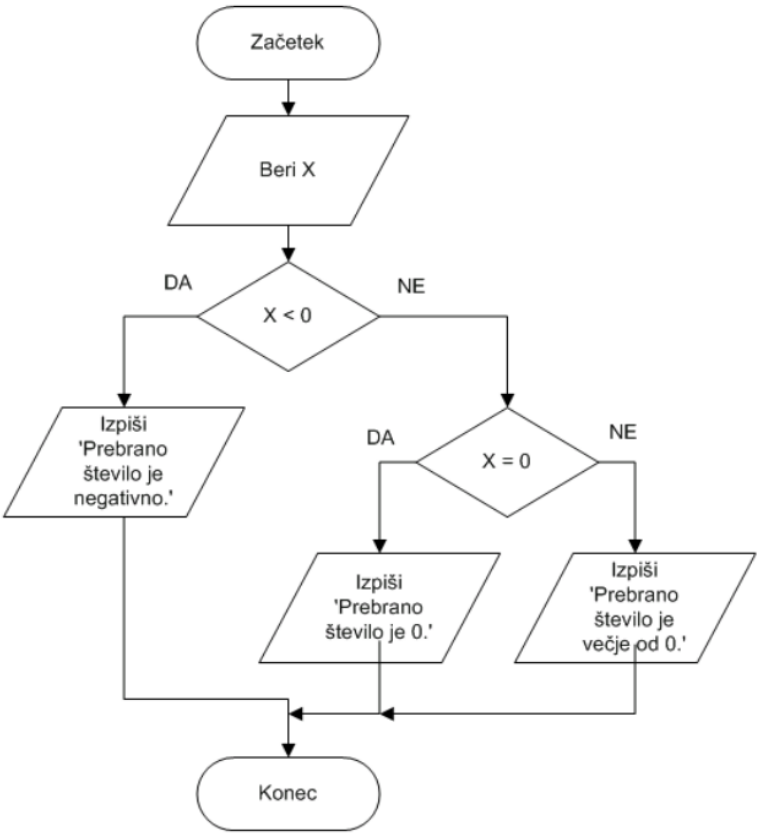
Diagram poteka je grafična predstavitev (ponazoritev) algoritma ali strukture, kjer so posamezne aktivnosti predstavljene z različnimi geometrijskimi liki, med seboj povezanimi z usmerjenimi povezavami, ki ponazarjajo potek izvajanja.

Diagram poteka (a*ngl. flowchart, flowsheet*) ponazarja pretok podatkov in informacij od začetka do konca programa s pomočjo grafičnih simbolov in povezavami. Uporaben je na vseh področjih dela, kjer je treba prikazati postopke, dogajanja, organizacijo ali odločitveno drevo. Diagram lahko služi za načrtovanje postopka, za njegovo dokumentiranje, kot opomnik pri izvedbi ipd. Pravila za sestavljanje diagrama so preprosta, zelo pomembno pa je, da nobena možna pot ali izbira ne ostane neizražena. Kot rečeno diagram poteka zasleduje tok informacij od začetka do konca programa. Za vsak ukaz je predviden grafični simbol določene oblike, v katerem je zapisan ukaz, medtem ko so povezave ponazorjene s puščicami, ki ponazarjajo tok izvajanja.

Tabela 1: Simboli diagrama poteka.

Oblike simbolov		**pomen**
	oval	začetek in konec algoritma
	paralelogram	vhodni in izhodni stavki
	pravokotnik	prireditev vrednosti konstanti ali spremenljivki, tudi rezultat izvajanja
	diamant	označuje odločitve, vejenje algoritma (IF)
	heksagon	začetek ponavljanja dela kode
	pravokotnik z dvojno črto	vključevanje zunanjih algoritmov, funkcij ali podprogramov
	puščice	povezava in smer izvajanja programa
	krog	kombiniranje toka izvajanja
	črtkane linije in odprti pravokotnik	dodajanje komentarjev

Kot primer diagrama poteka je predstavljen enostavni problem. Preberemo poljubno število in ugotovimo ali je večje od nič ali je manjše od nič ali enako nič.



Slika 1: Primer diagrama poteka.

1.2 Karakteristike računalnika

Računalnik ima kot stroj določene karakteristike, katerih razumevanje je bistveno:

Avtomatičnost: če računalniku predpišemo aktivnosti lahko dela brez intervencije uporabnika.
Hitrost: izvajanje procesiranja podatkov je izjemno hitro (meri se v mikrosekundah 10^{-6}, nanosekundah 10^{-9} in picosekundah 10^{-12}).
Natančnost: natančnost računalnika je konstantno visoka in rang natančnosti je odvisen od zasnove. Napake so posledica netočnih podatkov ali nezanesljivih programov (*angl. GarbageInGarbageOut*).
Zanesljivost: računalnik je stroj, nikoli ni utrujen od monotonega dela, nikoli mu ne pade koncentracija; Kontinuirano dela dokler ne zmanjka energije, se zgodijo katastrofe, se pregreje, ipd.
Vsestranskost: računalnik lahko izvede kakršnokoli nalogo, ki jo sestavlja končno število logičnih korakov.
Moč pomnjenja: računalnik lahko shrani in prikliče poljubno količino podatkov, saj lahko uporablja tudi sekundarne vire. Izguba ali pozabljivost podatkov je lahko le programirana.
Inteligentnost: računalnik izvaja samo stvari za katere je programiran in ne more izvajati lastnih zaključkov.
Čustva: računalnik nima emocij. Izvaja operacije in se odziva kot je sprogramiran.

1.3 Kratka zgodovina računalništva

Zgodovina računalnikov je sorazmerno kratka. Zgodovina razvoja računalnikov je opisana v mnogih učbenikih in spletnih straneh, zato je v tem poglavju povzeta zelo na kratko.

Razvoju mnogih tehnologij, kot so telefon ali žarnica, se da slediti do nastanka: kraju, času in posamezniku. Računalnik za razliko od mnogih tehnologij ni bil razvit v določeni sobi ob genialni ideji. Ideja, ki je vodila do razvoja računalnika je vključevala stoletja znanj in prispevkov mnogih ljudi, ki so s svojimi znanstvenimi dognanji omogočili razvoj računalnika.

Z razmahom osebnih računalnikov in Interneta je računalništvo prišlo v fazo široke rabe, ko večina uporabnikov uporablja računalnike brez poznavanja osnov delovanja in njihove zgradbe – podobno kot pri avtomobilih. Tako ob nakupu večina posluša nasvete poznavalcev ali trgovcev in ve le, da mora imeti čimveč gigahercov in megabajtov, od programov pa uporabljajo to, kar uporabljajo vsi drugi.

Grobo lahko razdelimo razvoj na naslednja obdobja:

Zgodnje obdobje: do 1940

Osnovo računalnika tvorijo znanstvene discipline: matematika, logika in računstvo, katerih osnove so postavili Grki, Egipčani, Babilonci, Indijci, Kitajci in Prezijci pred nekako 3000 leti.

1614 – Napier logaritmi, poenostavitev matematičnih izračunov;

1622 – iznajdba drsnih kalkulatorjev;

1672 – Pascal (seštevanje in odštevanje);

1674 – Leibnitz (seštevanje odštevanje, množenje in deljenje)

1801 – Jacquard (luknjane kartice)

1823 – Babbage (osnovne operacije na 6 decimalnih mest)

1830 – Babbage (zamisel analitskega stroja)

Rojstvo računalnika: 1940-1950

Razvoj elektronskega, splošnega računalnika (Mark I, ENIAC)

1946 - Von Neumann računalniškia arhitektura (skupno shranjevanje podatkov in programov v pomnilniku, EDVAC)

Uporaba računalnika: 1950 do danes

1950-59 – prva generacija, elektronke, velikost

1959-65 – druga generacij, tranzistorji, zmanjšanje velikosti in povečanje zanesljivosti, višjenivojski programski jeziki

1965-75 – tretja generacija, integrirana vezja, zmanjšanje velikosti in povečanje zanesljivosti, prvi mikroračunalnik, začetek programske industrije;

1975-85 – četrta generacija, zmanjšanje na velikost pisalnega stroja, PC, začetek računalniške mreže, elektronske pošte, GUI;

1985-... – peta generacija , paralelno procesiranje, grafika visoke ločljivosti, multimedija, integracija glkobalnih telekomunikacij, brezžične povezave, velike pomnilniške kapacitete;

1.4 Predstavitev podatkov v računalniku

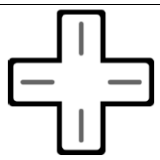
Računalnik je multimedijska naprava, kar pomeni, da je sposoben shranjevati in obdelovati heterogene podatke, a le v digitalni obliki. Vse podatke, ki so v naravi analogni je potrebno pretvoriti v digitalne. Podatki so lahko števila, znakovni nizi (besedila), zvok, slika in video.

Podatek je na formalen način predstavljeno dejstvo, koncept ali ukaz, ki mora biti primeren za komunikacijo, obdelavo in predstavitev tako človeku kot računalniku.

Informacija je nadgradnja podatka, saj mora biti podatek za informacijo postavljen v nek kontekst z določenim pomenom. Lastnosti informacije so, da je potrebno za njihovo interpretacijo predznanje in, da je to sporočilo med dvema uporabnikoma preko informacijskega medija. S pomočjo informacij človek nadgrajuje svoje znanje.

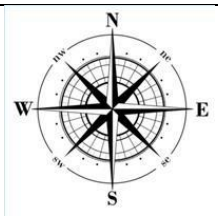
Podatek: 8,5 Informacija: Povprečna ocena študija je 8,5%.

Količino informacije lahko merimo za kar uporabljamo dvojiški številski sistem. Osnovna enota je bit (*angl. binary digit*), ki je najmanjša enota informacije, s katero lahko predstavimo dve stanji. Za prikazovanje več stanj uporabljamo niz bitov. Za opis štirih stanj potrebujemo 2 bita, za opis osmih stanj 3 bite, itn. Slika 2 prikazuje primera uporabe 2 in 3 bitov za opis stanj.



2 bita - 4 stanja

```
00 – gor
01 – dol
10 – levo
11 – desno
```



3 biti – 8 stanj

```
000 –sever
001 –jug
010 –vzhod
011 –zahod
100 –severovzhod
101 –severozahod
110 –jugovzhod
111 –jugozahod
```

Slika 2: Zapisi več stanj z bitnimi nizi.

Za zapis več stanj potrebujemo nize, ki jih združujemo v skupine po osem. Zaporedje ali niz ničel in enic imenujemo binarni niz. Tako niz osem bitov predstavlja en zlog in opisuje 256 stanj in se imenuje **byte**. Običajno je pri računalniškem zapisu besedil vsak znak shranjen v enem zlogu, za shranjevanje števil navadno uporabimo štiri ali osem zlogov, odvisno od velikosti števila in želene natančnosti zapisa. Ker v računalništvu uporabljamo dvojiški številski sistem, so tudi mnogokratniki enot prirejeni. Tako je kilo zlog (angl. Kilobyte - Kb) enak 2^{10} = 1024 zlogov.

Pri opisu večjih enot uporabljamo grške predpone. Med uporabniki velja dogovor da mali b označuje bit, veliki B pa byte.

- 1024 B = 2^{10} zlogov =1 kB (kilobyte)
- 1024 kB = 2^{20}($\approx 10^6$) zlogov = 1 MB (megabyte)
- 1024 MB = 2^{30}($\approx 10^9$) zlogov = 1 GB (gigabyte)
- 1024 GB = 2^{40}($\approx 10^{12}$) zlogov = 1 TB (terabyte)
- Peta=2^{50}, Exa=2^{60}, Zetta=2^{70}, Yotta=2^{80}

1.4.1 Predstavitev števil in številski sistemi

Številski sistem opisuje urejenost števil. Poznamo pozicijske in nepozicijski sisteme. Nepozicijski sistemi uporabljajo simbole, od katerih ima vsak določeno vrednost. Tipični predstavnik je sistem rimskih števil, ki uporablja 7 simbolov.

simbol	I	V	X	L	C	D	M
vrednost	1	5	10	50	100	500	1000

Za zapis števila moramo upoštevati pravila.

III	➔	1+1+1	=	3
IV	➔	5-1	=	4
VIII	➔	5+1+1+1	=	8
XVIII	➔	10+5+1+1+1	=	18
XIX	➔	10+(10-1)	=	19
LXXII	➔	50+10+10+1+1	=	72
CI	➔	100+1	=	101
MMXIII	➔	1000+1000+10+1+1+1	=	2013

Ker je zelo težko izvajati aritmetične operacije nepozicijskih številskih sistemov v računalništvu ne uporabljamo.

V vsakdanji uporabi uporabljamo pozicijski sistem ali sistem z mestnimi vrednostmi, v katerem je vsako število izraženo v obliki polinoma. Gradniki števila so števke ali cifre, s katerimi po polinomskem pravilu sestavljamo skupine, ki predstavljajo števila v izbranem številskem sistemu.

V vsakdanjem življenju smo vajeni desetiške predstavitve števil, medtem ko računalnik za zapis uporablja dvojiški sistem. Med obema sistemoma je potrebna konverzija. Konverzija med številskimi sistemi je enostavna.

Predstavitev celih števil

Predstavitev celih števil lahko imenujemo tudi predstavitev števil z fiksno decimalno piko, saj je pozicija decimalne pike znana in za njo so same ničle. Tako decimalno piko predpostavimo a je ne shranjujemo.

Najprej poglejmo predstavitev pozitivnih števil.

Desetiški sistem uporablja deset števk (0,1,2,3,4,5,6,7,8,9).

Število 1039 v desetiškem sistemu predstavimo kot:

$1024_{(10)} = 1 \cdot 1000 + 0 \cdot 100 + 2 \cdot 10 + 4 \cdot 1 = 1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$

Dvojiški sistem uporablja dve števki (0,1) in konverzija v desetiški sistem:

$101111_{(2)} = 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$

$= 1\cdot32 + 0\cdot16 + 1\cdot8 + 1\cdot4 + 1\cdot2 + 1\cdot1 = 47$

Za konverzijo iz desetiškega v dvojiškega pa uporabimo metodo deljenja in ostankov. Število delimo z osnovo, v tem primeru z 2, in v obratnem vrstnem redu zapišemo ostanke pri deljenju. Ostanki pri deljenju so lahko samo 0 in 1.

47 : 2 = 23 ostane **1** 23 : 2 = 11 ostane **1** 11 : 2 = 5 ostane **1** 5 : 2 = 2 ostane **1** 2 : 2 = 1 ostane **0** 1 : 2 = 0 ostane **1**	↑	ostanke prepišemo od spodaj navzgor in dobimo dvojiško predstavitev števila **$47_{(10)} = 101111_{(2)}$**

Z zapolnitev predstavitve v računalniku do polnega zloga dodamo v levo ustrezno število ničel.

$47_{(10)}$ 8 bitni zapis **$0010\ 1111_{(2)}$** 16 bitni zapis **$0000\ 0000\ \ 0010\ 1111_{(2)}$**

Če imamo na razpolago 8 bitov lahko zapišemo pozitivna cela števila v rangu 0÷255. Takšen sistem je enostaven in deluje za pozitivna cela števila. Za negativna števila se rang razdeli na dva enaka dela. Število predstavimo s sedmimi biti, prvi bit pa uporabimo za predznak. Če je prvi bit 0 je število pozitivno, če je 1 pa negativno.

$0010\ 1111_{(2)} = 47_{(10)}$; **$1010\ 1111_{(2)} = -47_{(10)}$**

Velika večina računalnikov uporablja za zapis negativnih števil dvojiški komplement. Rang števil je razdeljen na dva enako velika dela. Prvi del za pozitivna in drugi del za negativna števila.

Dvojiški komplement:

1. naredi inverz dvojiškega števila
2. prištej 1
3. rezultat je negativna vrednost prvotnega števila

$0000\ 1010_{(2)}=+5_{(10)}$

$1111\ 1011_{(2)}=-5_{(10)}$

Tako lahko predstavimo z binarnimi nizi nepredznačena in predznačena cela števila.

Tabela 2: Rangi celih števil.

	nepredznačena	**predznačena**
8 bitov	$0\div255 (=2^8-1)$	$-128\div127\ (-2^7\div2^7-1)$
16 bitov	$0\div65535 (=2^{16}-1)$	$-32768\div32767\ (-2^{15}\div2^{15}-1)$
32 bitov	$0\div4294967295\ (=2^{32}-1)$	$-2147483648\div2147483647\ (-2^{31}\div2^{31}-1)$

Predstavitev realnih števil

Realno število je sestavljeno iz treh delov: celega, decimalne pike in decimalnega dela. Predstavitev realnih števil se v računalniku izvede s pomočjo dveh celih števil. Uporablja se pomična vejica (*angl. floating point*), ki določa, da mora biti decimalna pika pred prvo števko, ki je različna od nič. Realna števila lahko zapišemo zelo različno z uporabo eksponenta.

$3.14 \quad 314\cdot10^{(-2)} \quad 0.00314\ 10^{(3)} \quad 0.314\ 10^{(1)}$

Standardni zapis števila z decimalno piko:

$0.xy\ e^n$, kjer so xy števke in x različen od 0; e–izbrana baza; n–eksponent

Na tak način lahko predstavimo realno število z dvema celima številoma: mantiso in eksponentom.

$0.\mathbf{314} \cdot 10^{\mathbf{1}}$; 314 je mantisa in 1 eksponent in zapis števila (**314,1**)

Seveda ne smemo pozabiti tudi na predznak. Natančnost zapisa je odvisna od velikosti binarnega niza znakov. Za zapis realnega števila običajno potrebujemo 32 bitov za dvojno natančnost (*angl. double precision*) pa 64 bitov. Po standardu IEEE so deli za zapis določeni po dolžini niza:

predznak	**eksponent**	**mantisa**	**skupaj**
1 bit	8 bitov	23 bitov	32 bitov
1 bit	11 bitov	52 bitov	64 bitov

Posebnost je ničla. Po tej metodologiji bi imeli lahko +0.0 in -0.0 zato velja dogovor da je ničla predstavljena s samimi ničlami (predznak, eksponent in mantisa).

1.4.2 Predstavitev znakov in znakovnih nizov

Pod znake razumemo vse črke abecede, ločila, številke, posebni znaki in kontrolni znaki. Posebni znaki so na primer:#,$,%,&,!,', pa tudi operatorji +,-,*,/. Kontrolni znaki, ki jih ne moremo natisniti (*angl. white spaces*), omogočajo vodenje izpisa ali aktivnosti (znak za konec vrstice, novo stran, *escape*). Vsi ti znaki so predstavljeni z binarnimi nizi in so standardizirani v kodnih tabelah. Najbolj znana je 7bitna predstavitev ameriškega standarda ASCII, ki jo uporabljajo vsi računalniki. V tej kodni tabeli so predstavljeni vsi znaki angleške abecede in posebni ter kontrolni znaki.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

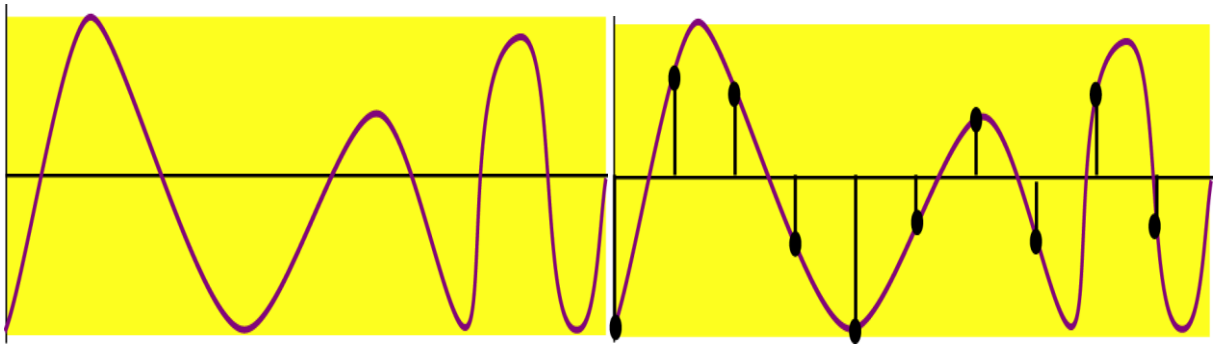
Source: www.LookupTables.com

Slika 3: Kodna tabela ASCII.

Smiselno bi bilo uporabiti en zlog, 8 bitov, s čemer lahko dodamo 128 znakov (npr: nemški ä slovenski š) in kodna tabela se imenuje ISO Latin 2. Zadnja standardna kodna tabela se imenuje Unicode, ki podpira zapis z enim, dvema in štirimi zlogi. To tabelo podpirajo vsi operacijski sistemi in vključuje tudi cirilico, arabsko, grško, turško, japonsko, … pisavo.

1.4.3 Predstavitev zvoka

Prestavitev zvoka se razlikuje od predstavitve števil ali znakovnih nizov. Zvok ni številna kategorija in je primer analognih podatkov. Četudi bi lahko merili vse njegove vrednosti bi potrebovali neskončen zapis, kar z računalnikom ni mogoče. Zato uporabljamo metodo vzorčenja.



Slika 4: Prikaz analognega signala v časovnem koraku in vzorčenje signala..

Vzorčenje pomeni, da izberemo samo končno število točk na analognem signalu, izmerimo njihovo realno vrednost in jih shranimo kot cela števila (zaokrožanje).

Shranjene vrednosti so kot bitna globina (*angl. bit depth*) za vsak vzorec in število vzorcev na sekundo. Produkt teh dveh imenujemo tudi (*angl. bit rate*). Tako za 40000 vzorcev in 16 biti na vzorec dobimo zapis 640000bitov na sekundo. Danes najbolj znani zapis je MP3 (MPEG Layer 3), ki pa je že komprimiran zapis glasbe. Uporablja 44100 vzorcev na sekundo pri 16 bitnem vzorcu.

1.4.4 Predstavitev slik

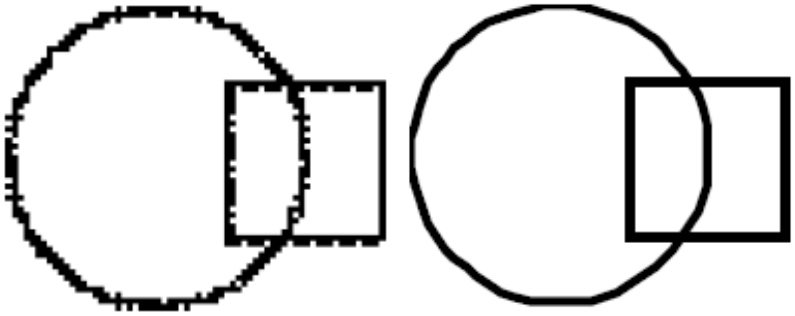
V računalniku v osnovi delimo dva tipa predstavitve slik: rastrsko (bitno) in vektorsko računalniško grafiko.

Rastrska grafika je shranjevanje analognih podatkov (fotografije) podobno kot zvok, s tem, da shranjujemo vzorce v prostoru in ne v času kot pri zvoku. Vzorčenje imenujemo skeniranje (*angl. scanning*) in vzorci se imenujejo grafične točke. Rastersko grafiko sestavlja matrika, polje pik imenovanih grafične točke (*angl. pixl*). Gostoto zapisa merimo s številom točk na dolžinsko enoto (*angl. dots per inch*) in jo imenujemo ločljivost (*angl. resolution*), število bitov za prikaz vrednosti pa barvna globina (*angl. colour depth*). Grafična točka je najmanjši del, ki je programsko dosegljiv in mu lahko definiramo barvo. Barva je shranjena kot bitni vzorec in število bitov določa število barv. Za zaslon se večinoma uporablja RGB model, kjer z tremi osnovnimi barvami (Red, Green, Blue) kombiniramo ostale barve. Vsaka barva je v najpogostejšem 24 bitnem RGB modelu predstavljena z enim zlogom. V desetiškem sistemu je predstavitev:

```
(0,0,0) črna, (255,255,255) bela, (255,0,0) rdeča, (0,0,255) modra;
```

Predstavitev omogoča kakovostno predstavitev slik, naravni videz in enostavno tiskanje. Med slabosti pa štejemo predvsem veliko porabo pomnilniškega prostora in težave pri spreminjanju. Zasedanje veliko prostora rešujemo z metodami stiskanja, komprimiranja. Najpogostejša metoda je JPG.

V vektorski grafiki je slika opisana z osnovnimi gradniki, npr: črte in krivulje, ki so opisani z matematičnimi formulami in besedilom v določenem fontu. Slike so manj naravne, a jih lahko hitro spreminjamo brez vpliva na kakovost in zasedajo manj prostora.



Slika 5: Rastrska in vektorska grafika.

1.4.5 Predstavitev videa

Video ali film je predstavitev slik v časovnih okvirjih (*angl. frames)*. Video je torej predstavitev podatkov v prostoru in času. Če zmoremo vsakega posebej, lahko tudi kombinacijo.

2 ARHITEKTURA RAČUNALNIKA

O arhitekturi ali zgradbi računalnika je na voljo obilo knjig in elektronskih virov. Razvoj računalništva nasploh je izredno hiter, tako da so današnji podatki jutri že stari. V tem poglavju je opisana osnovna arhitektura računalnika, ki se veliko ne spreminja. Za poglobljeno znanje tega področja je priporočeno brskanje po literaturi.

Slika 6: Gorazdov osebni računalnik.

Večina ljudi besedi računalnik pomisli na škatlo z nekaj gumbi, ki je z množico kablov povezana z zaslonom, tipkovnico in miško ter običajno še s tiskalnikom in zvočniki. Računalniki so danes izredno domiselno oblikovani.

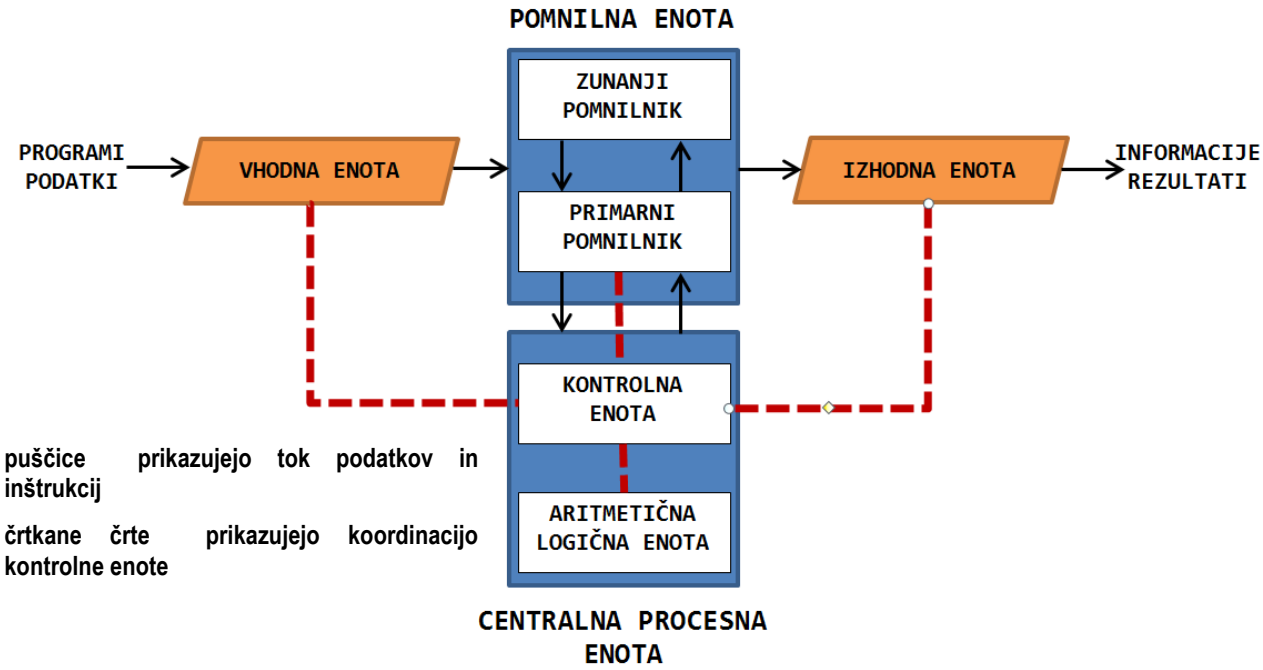
Za učinkovito rabo računalnika je potrebno poznavanje vsaj osnovnih komponent in zgradbe računalnika.

2.1 Zgradba in delovanje računalnika

Računalniki ali tudi računalniški sistemi so stroji za obdelavo podatkov sestavljeni iz:
strojne opreme (*angl. hardware*),
programske opreme (*angl. software*) in
komunikacijske opreme.

Pri definiciji termina »računalnik« se običajno opiramo na pojem »von Neumannov« računalnik. Ta definira računalnik kot stroj, ki je sestavljen iz štirih osnovnih delov (slika 8):

- centralne procesne enote (*angl. central process unit - CPU*), imenovane processor, ki je setavljen iz dveh delov:
  - aritmetične logične enote, kjer se izvajajo instrukcije med procesiranjem
  - kontrolne enote, ki koordinira delo vseh ostalih enot.
- pomnilniške enote
  ki shranjuje podatke in instrukcije potrebne za procesiranje iz vhodne enote, vmesne rezultate procesiranja in rezultate procesiranja, preden so poslani na izhodno enoto.
  ki je sestavljen iz dveh delov: notranjega, ki je hiter, omejen po kapaciteti, drag, ter izgubi podatke, ko ostane brez energije in zunanjega, ki je počašnejši, cenejši, velikih kapacitet in ohranja podatke tudi ko ostane brez energije.
- vhodne enote: sprejema instrukcije in podatke iz zunanjega sveta, jih prevede v računalniku sprejemljivo obliko in jih posreduje računalniku v procesiranje.
- izhodne enote: sprejme rezultate procesiranja računalnika, jih preoblikuje v človeku razumljivo obliko in posreduje v zunanji svet.



Slika 7: Osnovna organizacija računalnika.

2.1.1 Osnove delovanja

Centralna procesna enota – **CPE** iz centralnega pomnilnika jemlje ukaze in jih izvršuje. Običajno jo delimo na: krmilno - kontrolno enoto, aritmetično-logično enoto in registre. Za povezave med njimi skrbijo notranje podatkovne poti, za povezavo procesorja s pomnilnikom in vhodno/izhodnim sistemom pa zunanje podatkovne poti.

Delovanje računalnika je določeno z ukazi, ki jih CPE jemlje iz pomnilnika enega za drugim. Vsak ukaz je sestavljen iz dveh korakov:

Jemanje ukaza iz pomnilnika (angl. fetch). Krmilna enota analizira tip operacije in ustrezno nastavi stikala aritmetično-logične enote; nato izračuna naslov operanda v pomnilniku in vzpostavi zvezo z njim.

Aritmetično-logična enota prebere podatek iz pomnilnika in opravi programirano operacijo.

Centralni pomnilnik je elektronsko vezje, ki hrani podatke in programe. Ukazi v pomnilniku so elementarni, zapisani dvojiško – binarno. Imenujemo jih tudi strojni ukazi (*angl. machine code*). Zaporedje ukazov, ki izvaja neko nalogo imenujemo program. Naslov ukaza v pomnilniku je zapisan v programskem števcu (*angl. program counter*). Po vklopu računalnika se v programski števec prenese naslov prvega ukaza. CPE po vsakem izvedenem ukazu vrednost števca poveča za 1, razen pri ti. skokih, pri katerih se v števec zapiše naslov novega ukaza, ali pa pri prekinitvah (*angl. interrupt*). Pri prekinitvi se vrednost programskega števca shrani, izvede pa se prekinitveni servisni program. Po končani prekinitvi se izvajanje programa nadaljuje na shranjenem naslovu.

Dvojiški zapis

Ker je z uporabo elektronike lažje opisovanje dvojiških stanj, uporabljamo pri elektronskih računalnikih dvojiški – binarni sistem. Osnova količina informacije je binarno število bit, ki lahko opiše le dve različni stanji. Običajno jih ponazarjamo s števili 0 in 1. Za opisovanje večjih količin informacij bite združujemo v skupine, običajno po 8, 16, 32 ali 64.

Enota za podajanje velikosti pomnilnika je zlog ali bajt (angl. byte), ki je sestavljen iz osmih bitov. Zlog lahko opiše $2^8 = 256$ različnih stanj, torej 256 različnih števil ali znakov.

2.1.2 Značilnosti CPE, pomnilnika in prenosnih poti

Centralne procesne enote

oz. procesorji so danes večinoma izdelani na eni rezini – čipu (*angl. chip*). Starejši procesorji oz. procesne enote večjih računalnikov pa so lahko izvedene tudi kot integrirana vezja z več elementi – čipi.

- Zmogljivost procesorja je odvisna od velikosti registrov in širine notranjih podatkovnih poti v procesorju. Tako poznamo 8, 16, 32, 64 in več bitne procesorje. Večja širina pomeni hitrejše delovanje, saj je mogoče npr. sešteti dve 32 bitni števili naenkrat, brez delnih vsot in prenašanja. Ker imajo nekateri procesorji različno širino notranjih in zunanjih podatkovnih poti in velikosti registrov, pri oznaki procesorja podamo najmanjšo vrednost.
- Hitrost delovanja procesorja je odvisna od takta, v katerem se izvajajo ukazi in prenosi podatkov. Merimo ga v MHz ali GHz.
- Glede na zahtevnost ukazov, ki jih procesorji izvajajo ločimo:
  - CISC - (Complex Instruction Set Computer) so procesorji, ki obvladajo veliko število zelo zahtevnih ukazov, izvajanje posameznih ukazov traja več ciklov.
  - RISC - (Reduced Instruction Set Computer) so procesorji, ki obvladajo samo omejen nabor osnovnih ukazov, zato pa jih izvajajo optimalno hitro - večinoma le v enem ciklu.

Pogosto zmogljivosti računalnikov povečujejo tudi pomožni procesorji ali koprocesorji. To so procesorji, izdelani za posebne naloge, npr. za operacije z decimalnimi števili ali za grafične operacije. Takšni koprocesorji razbremenijo glavni procesor in pospešijo delovanje računalnika. Dandanes so razširjeni specializirani grafični procesorji, ki so pogosto tudi zmogljivejši od centralnih procesorjev.
Hitrost obdelave podajamo s številom izvedenih ukazov v sekundi (*angl. million instructions per second - MIPS*). Odvisna je od hitrosti dela procesorja – takta, in od zapletenosti ukazov. Primer: procesor s taktom 500 MHz, ki izvaja vse ukaze v enem ciklu - temu idealu se približajo procesorji RISC - bi dosegel hitrost 500 MIPS.
Število izvedenih operacij z decimalnimi števili (angl. floating point operations per second - FLOPS) podaja predvsem hitrost matematičnega soprocesorja oz. hitrost 'računanja', ki je zanimiva pri numerično zahtevnih problemih. Obe enoti sta primerljivi le za isti tip procesorja, zato se za primerjavo hitrosti različnih procesorjev uporabljajo testi - programi s katerimi merimo hitrost.

Pri poslovnih aplikacijah računske operacije niso zahtevne, zato pa je potrebno čim hitreje obdelati velike količine podatkov, zato je pomembnejša hitrost shranjevanja in dostopa do podatkov.
Vsi podatki o hitrostih so relativni in jih je potrebno upoštevati z rezervo, saj so odvisni od izvedbe procesorja in uporabljenega programskega jezika. Za realno primerjavo je potrebno na različnih računalnikih primerjati programe, ki jih želimo uporabljati v normalni uporabi, oz. v revijah poiskati primerjalne teste, ki se najbolj približajo našim zahtevam npr. numerično zahtevni programi ali velik pretok podatkov.

Pomnilnik

Za delo računalnik uporablja bralno-pisalni pomnilnik s poljubnim dostopom *(angl. Random Access Memory - RAM*), v katerega naloži program, nato pa med delom vanj vpisuje podatke in iz njega bere. Večinoma se uporablja »dinamični RAM« - DRAM, ki je cenejši, vendar pa potrebuje posebno vezje za obnavljanje vsebine, sicer se vsebina izgubi. »Statični RAM« - SRAM - ne potrebuje obnavljanja, ima manjšo porabo in je hitrejši, vendar tudi dražji. Zato se uporablja v manjših pomnilnikih, ali tam, kjer je potrebna večja hitrost. Po izklopu napajanja se vsebina pomnilnika izgubi. Zaradi majhne porabe lahko pri statičnem pomnilniku vsebino

ohranimo z minimalnim napajanjem – običajno baterija ali kondenzator – tudi po izklopu računalnika. Razvoju procesorjev sledi tudi razvoj pomnilnikov. Za hitrejši prenos podatkov se uporablja dvojni takt prenosa (angl. Double Data Rate – DDR), z miniaturizacijo pa se povečujejo njihove zmogljivosti.
Za podatke in del programa, ki ga računalnik potrebuje ob zagonu, in mora torej ostati nespremenjen tudi med izklopom računalnika, uporabljamo bralni pomnilnik (*angl. Read Only Memory- ROM*). To je vezje, v katerem je tovarniško zapisana vsebina, ki jo računalnik lahko prebere, ne more pa je spremeniti. Običajno ROM predstavlja manjši del skupnega pomnilnika.
Velikost pomnilnika, ki ga računalnik lahko uporablja, je odvisna tudi od števila bitov, s katerim je naslov podan. Če je npr. naslovna beseda dolga 24 bitov, je naslovni prostor – to je največji možni pomnilnik – velik 2^{24} = 16Mb, 32 bitni procesorji pa lahko naslovijo 4Gb.
Na hitrost delovanja računalnika vpliva tudi hitrost dostopa do podatkov v delovnem pomnilniku, ki mora biti usklajena s hitrostjo procesorja, sicer pride do čakalnih stanj. Dostopni čas pomnilnika podajamo v nano sekundah - 10^{-9}s ali s frekvenco, s katero lahko procesor prenaša podatke. Zato tudi obstajajo različne vrste RAM, npr.: SDRAM, DDR ali RIMM, kar povzroča težave predvsem pri izbiri komponent računalnika in kasnejših nadgradnjah.

Predpomnilnik (angl. cache)

Računalniki s počasnim pomnilnikom imajo dodan hiter predpomnilnik, običajno izveden kot statični RAM, ki premosti razliko v hitrosti med procesorjem in pomnilnikom. Novejši mikroprocesorji imajo že na samem čipu vgrajen predpomnilnik.

Prenosne poti

Za prenose podatkov med elementi računalnika uporabljamo prenosne poti. Te so lahko izvedene kot povezava točka-v-točko (*angl. point-to-point*), v tem primeru povezujejo samo dva elementa, ali pa kot vodilo (*angl. bus*), priklop več enot na skupno prenosno pot preko odcepov. Prenosi so izvedeni kot:

- *vzporedni – paralelni.* Ta prenos potrebuje več žilno povezavo – za vsak bit po eno žico. Hitrost prenosa je odvisna od širine poti – 8, 16, 32, 64 ali več bitov, ter od frekvence podane v MHz. Zaradi večjega števila žic in visoke frekvence so omejeni na krajše razdalje in se uporabljajo predvsem znotraj procesorja oz. računalnika.
- *zaporedni – serijski.* Pri tem so posamezni biti pretvorjeni v zaporedje, ki ga pošljemo po eni povezovalni liniji in na sprejemni strani spret pretvorimo v začetno obliko.

Vodilo

Večina računalnikov uporablja za prenose skupno - sistemsko vodilo (*angl. system bus*), na katero so priključene vse komponente. Fizično je vodilo izvedeno kot množica povezovalnih linij, ki prenašajo električne signale, tam kjer je priključena enota, pa je odcep. Vodilo prenaša naslovne signale, ki določajo lokacijo v pomnilniku, podatkovne signale za prenos podatkov in nadzorne, kontrolne signale za sinhronizacijo posameznih komponent računalnika. Od širine in hitrosti vodila je odvisna »izkoriščenost« procesorja, s tem pa tudi hitrost delovanja računalnika. Širino podajamo v bitih in vpliva na količino hkratnega prenosa podatkov, hitrost prenosa pa določa takt vodila, ki ga podajamo v MHz. Ker je na vodilo priključenih več naprav, le-to predstavlja ozko grlo, saj lahko po njem podatke hkrati prenaša samo ena naprava, hitrost pa je omejena s hitrostjo najpočasnejše med njimi. Zato mnogi računalniki uporabljajo več prenosnih poti in tako omogočijo istočasen prenos podatkov med več napravami hkrati. Različna vodila povezuje most (*angl. bridge*).

Osebni računalniki uporabljajo standardna 32(64) bitna vodila **PCI** s hitrostjo do 33(66)MHz, ki omogočajo prenose podatkov je do 133,3(533,3) Mb/s.

Zmogljiva grafika potrebuje hitrejše prenose, zato za priključitev grafičnih kartic uporabljajo posebna 32 bitna vodila za grafiko (*angl. Advanced Graphics Port, AGP*), s hitrostjo 66MHz in ustreznimi mnogokratniki.

Oboje je v zadnjem času zamenjalo vodilo PCI-Express, ki ni paralelno temveč serijsko, sestavljeno iz več poti z dvosmernim prenosom po 2,5Gb/s, skupaj do 200Gb/s.

Večprocesorski računalniki

Zmogljivost računalnika je odvisna od zmogljivosti in števila procesorjev. Glede na število procesorjev ločimo

- eno procesorske računalnike, ki so v večini. Ti izvajajo en ukaz z enim podatkom (*angl. Single Instruction stream, Single Data stream – SISD*)
- več procesorske računalnike – paralelne, ki lahko izvajajo več ukazov z več podatki hkrati (*angl. Multiple Instruction stream, Multiple Data stream – MIMD*)
  - Nekaj procesorski imajo večinoma 2, 4, 8 ali 16 procesnih enot. Običajno vsaka CPE izvaja svoj program in med seboj skoraj ne komunicirajo. Lahko jih opišemo kot več paralelno delujočih računalnikov SISD.
  - Pravi MIMD računalniki imajo do več tisoč tesno povezanih procesorjev – vsak npr. s šestimi sosednjimi – ki sodelujejo pri reševanju istega problema.
- vektorske računalnike, ki izvajajo en ukaz z več podatki hkrati (*angl. Single Instruction stream, Multiple Data stream – SIMD*)

2.1.3 Zunanje - periferne naprave

Za smiselno delovanje računalnika centralna procesna enota ne zadošča. Potrebujemo še zunanje naprave in sicer trajni, masovni pomnilnik za shranjevanje programov in podatkov v času, ko računalnik ne deluje, ter vhodno/izhodne naprave za komunikacijo z uporabnikom.

Zunanje naprave priključimo na vodilo preko tako imenovanega krmilnika naprave (*angl. device controller*), ki omogoča prenos podatkov v napravo ali iz nje. Prenos podatkov lahko poteka preko procesorja (*angl. programmed I/O*) ali pa neposredno v pomnilnik (*angl. Direct Memory Access – DMA*). Slednji način je hitrejši, vendar je zaradi posebnega krmilnika DMA tudi dražji.

Trajni pomnilniki

Trajni ali zunanji pomnilniki so pomnilni mediji, ki lahko hranijo podatke tudi po izklopu napetosti. Izraz zunanji se je uveljavil v času, ko so bili ti pomnilniki dejansko fizično ločeni od osrednje enote. Danes so trajni pomnilniki večinoma vgrajeni v skupno ohišje s procesno enoto. Omogočajo poceni in trajno shranjevanje večjih količin podatkov, zato jih včasih poimenujejo tudi masovni pomnilniki. So približno sto krat cenejši od delovnega pomnilnika, vendar tudi tisoč krat počasnejši. Trajne pomnilnike lahko ločimo na:

- Fiksne, pomnilni medij in pogonska enota sta celota npr. disk. To omogoča hitrejše in zanesljivejše delovanje, vendar je količina podatkov omejena.
- Izmenljive, pomnilni medij in bralno-pogonska enota sta ločeni, npr. disketa in disketnik, tako da lahko v isti pogonski enoti zamenjamo pomnilni medij, ki je večinoma tudi cenejši. To omogoča shranjevanje večje količine podatkov, pri standardnih pomnilnih enotah pa tudi prenos podatkov na druge računalnike. Izmenljivi pomnilniki so običajno počasnejši, manjših zmogljivosti in manj zanesljivi.

Način zapisa pri pomnilniku je lahko:

- Magnetni, nosilec pomnilnega medija je prevlečen s tanko magnetno plastjo, v katero glave zapisujejo in berejo podatke. Pomnilni medij omogoča poljubno branje in pisanje, npr. diski, trakovi.

- Svetlobni, pisanje in branje podatkov s pomočjo laserja. Ker je za pisanje potrebna večja jakost žarka, se uporabljajo predvsem kot bralni pomnilniki - CD ROM.
- Elektronski (tranzistorski), posebna izvedba bralnega pomnilnika, ki omogoča tudi programiranje in elektronsko brisanje (*angl. Flash Memory*). Uveljavlja se predvsem kot prenosni pomnilni medij brez gibljivih delov, zmogljivosti do nekaj GB, ki se priključi na vodilo USB ali kot pomnilne kartice. V novejšem času nadomešča diskovne enote.

Diski

Najpogostejši trajni pomnilnik v današnjih računalnikih je disk (*angl. Hard Disc - HD*), saj le redko srečamo računalnike brez njih. Disk je aluminijeva krožna plošča, na obeh straneh prevlečena z feromagnetno plastjo. Informacije berejo in zapisujejo bralno/pisalne glave, ki se premikajo v radialni smeri, medtem ko se disk vrti. Za vsako magnetno plast je po ena glava. Podatki so zapisani v koncentričnih krogih, sledeh (*angl. track*), saj se glave ne premikajo zvezno ampak po korakih. To omogoča sorazmerno hiter, predvsem pa neposreden dostop do podatkov. Vsaka sled je razdeljena na več sektorjev. Za povečanje zmogljivosti je v disku združenih več plošč, sledi z enakim polmerom pa tvorijo cilinder. Ker je razdalja med glavo in namagnetenim slojem na plošči zelo majhna, so diski nepredušno zaprti, saj bi vsaka nečistoča poškodovala magnetni sloj in glavo.

Velikosti plošč, diskov so danes večinoma 3,5 in 2,5 palca, v prenosnih računalnikih 2 palca. Hitrosti vrtenja plošč so od 3600 do 10000 obratov v minuti, kar omogoča povprečne dostopne čase pod 10 mili sekund. Hitrost prenosa podatkov je odvisna od vodila oz. krmilne elektronike in se podaja v MB/s.

Za povečevanje zmogljivosti nekateri računalniki uporabljajo več diskov hkrati. Kadar želimo povečati tudi varnost in zanesljivost, uporabljamo polja diskov (*angl. Redundant Array of Inexpensive Disks - RAID*), kjer so podatki porazdeljeni in podvojeni na različnih diskih.

Diski so večinoma fiksni, starejše izvedbe so omogočale zamenjavo diskov v diskovnem pogonu. Danes obstajajo izvedbe izmenljivih diskov, ki v posebnem ohišju omogočajo priklop oz. zamenjavo celotnega diska s pogonom vred. Primerni so za prenašanje velikih količin podatkov ali za enostavno varovanje podatkov, saj lahko po delu celoten disk s podatki in programi odstranimo in shranimo na varno mesto.

Optične enote

Kmalu po uveljavitvi kompaktnih - laserskih diskov (*angl. compact disc - CD*) v glasbi, so le-te uporabili tudi v računalništvu. Prednost je v digitalnem zapisu, veliki zmogljivosti in razviti tehnologiji ter s tem nizki ceni. Ker je pri svetlobnem zapisu podatkov za branje potrebna manjša jakost žarka kot za pisanje, so se prvotno uveljavile bralne enote - **CD ROM**. Najbolj razširjene so plošče premera 12 cm in zmogljivosti do 700 Mb. Pri množični izdelavi je tudi cena ene plošče - zgoščenke nizka, zato so se uveljavile predvsem kot medij za distribucijo programske opreme.
Enote **CD-RW** omogočajo arhiviranje podatkov in izdelavo zgoščenk v manjših količinah. Običajno se uporabljajo cenejše laserske plošče z možnostjo enkratnega zapisa CD-R (*angl. CD - Recordable*). Enote za zapisovanje omogočajo tudi branje plošč. DVD (*angl. Digital Versatile Disk*), omogoča dvostransko in dvoslojno zapisovanje na disk z večjo gostoto zapisa in zmogljivostjo.

Vhodno / izhodne naprave

Računalnik shranjuje in obdeluje podatke samo v elektronski obliki. Za komunikacijo uporabnika z računalnikom zato potrebujemo naprave, ki te podatke pretvarjajo v uporabniku razumljivo obliko, npr. tekst, sliko ali zvok. Vhodne naprave uporabljamo za prenos informacij iz zunanjega sveta v računalnik npr. za posredovanje podatkov in navodil računalniku. Izhodne

naprave služijo obratnemu namenu, prenosu informacij iz računalnika v zunanji svet, npr. za prikaz rezultatov ali obvestil računalnika. Te naprave so pomembne zlasti pri trajni in množični uporabi računalnika, saj je prav od njih odvisno udobje in enostavnost dela z računalnikom.
Danes je večina računalnikov opremljenih z grafičnim zaslonom, tipkovnico in miško, glede na namen uporabe pa še z ustreznim tiskalnikom. Večina računalnikov ima dodane tudi zvočnike. Večji računalniški sistemi imajo vhodno/izhodni sistem, ki omogoča priključitev večjega števila uporabnikov hkrati.
Vhodno izhodne naprave lahko razvrstimo po več kriterijih – glede na način uporabe in glede na vrsto podatkov, ki jih posredujejo.

Danes vhodno/izhodne naprave, glede na vrsto podatkov ki jih posredujejo, delimo na :

- **tekstne** ali alfanumerične, ki omogočajo posredovanje besedila, številk in nekaterih znakov
- **slikovne (grafične),** ki prikazujejo mirujočo ali gibljive slike, oz. omogočajo vnos položaja in izbiro slikovnih elementov. Ker v sliki lahko prikazujemo tudi tekst, so grafični prikazovalniki povsem nadomestili tekstne.
- **zvočne** za zajemanje ali predvajanje zvoka.

Osnovne vhodno/izhodne naprave

Grafični zaslon in grafični procesor: Za prikazovanje teksta in slike danes računalniki uporabljajo kombinacijo grafične karice in zaslona. Grafična kartica skrbi za pretvorbo podatkov v obliko, ki jo zaslona prikazuje. Tako je od grafičnega procesorja odvisna predvsem zmogljivost in hitrost prikaza, od zaslona pa velikost in kvaliteta slike. Široka uporaba računalniške grafike je pospešila razvoj grafičnih zaslonov, tako da danes uporabljamo različne tehnologije za prikaz slike. V zadnjem času je intenziven razvoj zaslonov na dotik (angl. touchscreen), ki je posledica razvoja za telefone in uporabo, kjer je otežena uporaba miške. Izpostavimo lahko neergonomičnost uporabe.

Slika 8: Zasloni na dotik.

Tipkovnica je najbolj razširjena vhodna naprava, ki je priključena na skoraj vsak računalnik. Danes ima večinoma standardno obliko, s približno stotimi tipkami. Novejše tipkovnice vključujejo še posebne tipke, za hitro aktiviranje nastavljenih programov. Namenjene so predvsem enostavnejši uporabi Interneta, lahko pa vključujejo tudi nastavitve glasnosti in izklop zvoka.

Miška je krmilna naprava, ki je zadnje čase nepogrešljiva pri osebnih računalnikih in grafičnih postajah. Miška se je uveljavila predvsem zaradi enostavnega rokovanja, saj roka med delom počiva na podlagi.

Računalnik potrebuje za predvajanje zvoka zvočni sistem, sestavljen iz zvočnega procesorja in zvočnikov. Zvočni procesor je večinoma del osnovne plošče.

Tiskalniki omogočajo izpis podatkov na trajni medij v obliki teksta ali slike.

2.1.4 Povezave

Povezave omogočajo povezovanje komponent računalnika in računalnikov med seboj. Vsaka povezava je izvedena iz strojne opreme, to je fizične povezave med enotami in programske opreme, ki skrbi za prenos podatkov in odpravljanje motenj. Za uspešno povezovanje in prenašanje podatkov so dogovorjeni vmesniki – protokoli, ki predpisujejo način povezovanja. Izvedba povezave je odvisna od namena povezave, priključenih komponent in seveda od okoliščin, kot so oddaljenost, hitrost in cena.

Vzporedna - paralelna povezava

pri kateri prenašamo ves podatek hkrati, torej po 8, 16, 32 ali več bitov naenkrat, potrebujemo pa seveda vsaj toliko povezovalnih linij, kot je podatkovnih bitov. Zaradi višje cene fizične povezave – večžilni kabel in omejene razdalje jo uporabljamo predvsem za povezave znotraj računalnika, ali pa za priključitev perifernih naprav, npr. tiskalnika. Uporabimo jo lahko tudi za povezovanje dveh računalnikov na krajši razdalji. Omogočajo prenose podatkov na krajših razdaljah do nekaj metrov.

Zaporedna - serijska povezava

Namesto pošiljanja podatkov po mnogih žicah, jih pri serijski povezavi pretvorimo v zaporedje impulzov, kjer vsak predstavlja en bit. Ta niz pošljemo po žici in jih na sprejemnem koncu znova pretvorimo v prvotno obliko. Univerzalno serijsko vodilo (*angl. Universal Serial Bus – USB*) je standard, za povezovanje zunanjih naprav, ki omogoča povezavo do 127 naprav in prenose do 12 MB/s (USB2 – 480MB/s). Zaradi višjih hitrosti sodobnih serijskih povezav, le-te nadomeščajo tudi paralelne povezave znotraj računalnika, npr. vodilo PCI-Express, SerialATA za priklop diskov, ter USB in FireWire za priklop hitrih zunanjih naprav.

2.2 Vrste računalnikov

Iz opisanih elementov lahko sestavimo računalnike poljubne sestave - konfiguracije. Minimalna konfiguracija zahteva eno centralno procesno enoto z nekaj pomnilnika, eno vhodno in eno izhodno napravo, vendar pa imajo danes vsi računalniki vsaj še eno zunanjo pomnilno enoto. Različne konfiguracije računalnikov so sestavljene glede na potrebe in možnosti uporabnikov, v grobem pa jih lahko razdelimo v nekaj skupin. Večina uporabnikov bo najpogosteje srečala

Osebni računalnik (*angl. personal computer*) so večinoma v namizni izvedbi (*angl. desktop*) ali v obliki majhnega ali večjega stolpiča (*angl. mini, midi, big tower*). Zelo so razširjeni prenosni računalniki, danes pretežno v obliki elektronske beležnice (*angl. notebook*) ali tablični računalniki (*angl. tablet*).

Slika 9: Osebni računalniki.

Delovna postaja (*angl. workstation*) je zmogljiv računalniški sistem namenjeni zahtevnejšemu delu, predvsem z uporabo računalniške grafike. Od osebnih računalnikov se ločijo po bolj zaprti in optimirani arhitekturi, ki jo sestavljajo hitre in kakovostne komponente (tudi gaming PC).

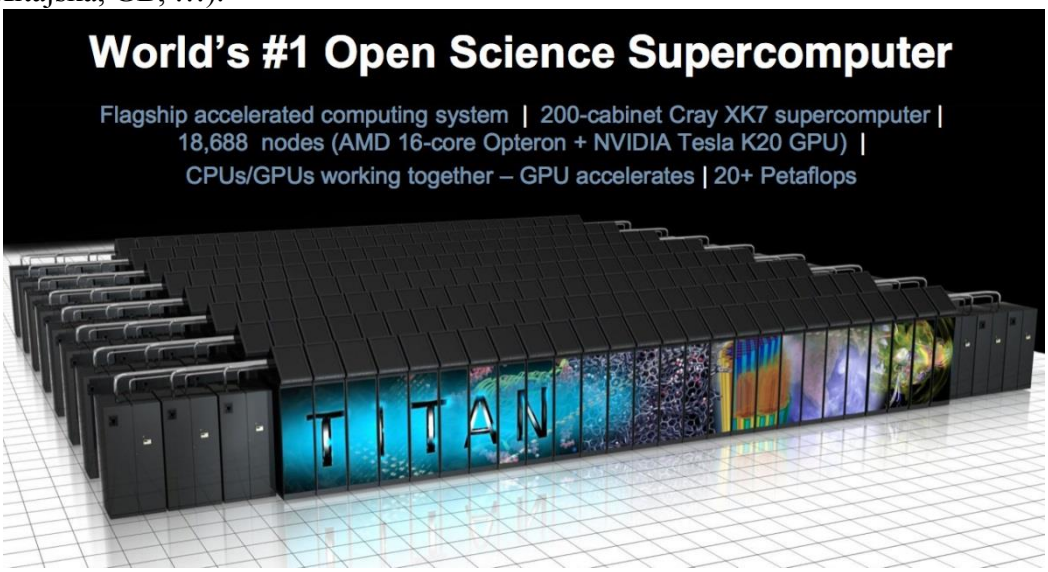
Zaradi hitrega razvoja osebni računalniki po zmogljivostih dosegajo delovne postaje, zato je meja velikokrat zabrisana.

Slika 10: Delovne postaje.

Strežnik (angl. server) je računalnik ali sistem namenjen predvsem skladiščenju in obdelavi podatkov. Poudarek je na pomnilnih zmogljivostih in pretoku podatkov, hitrih povezavah in zmogljivih procesorjih. Zato uporabljajo polja diskov za večje zmogljivosti ter podvojene procesorje za večjo zanesljivost Vhodno izhodne enote so namenjene samo vzdrževanju, saj do njihovih zmogljivosti uporabniki dostopajo preko omrežja.

Slika 11: Strežnik.

Superračunalniki (*angl. supercomputer*) so večprocesorski računalniki vrhunskih zmogljivosti, namenjeni predvsem za računsko intenzivno znanstveno in raziskovalno delo. Še pred leti so prevladovali vektorski računalniki izdelani z uporabo najnovejših tehnologij, npr. z uporabo superprevodnosti, elementov iz galijevega arzenida in optičnih elementov. Danes prevladujejo računalniki izdelani iz procesorjev - običajno RISC - dostopnih na trgu. Skupno število procesorjev je lahko tudi več tisoč. Zaradi visoke cene jih uporabljajo predvsem vladne organizacije in večja podjetja ali univerze in nekako sodijo v področje prestiža med državami (USA, Kitajska, GB, …).



Slika 12: Titan izvede 17.59 tisoč trilionov kalkulacij v sekundi. Uporablja se predvsem za izračune klimatskih sprememb, vremena, simulacije jedrskih tehnologij ipd.

3 PROGRAMJE

3.1 Operacijski sistem in programska oprema

Bistveni del računalnika, poleg strojne opreme, je programska oprema ali programje (angl. software). Programi so različni, od majhnih pomožnih programčkov za preračun temperature iz Celzijevih stopinj v stopinje Farnheita, do velikih programskih sistemov za napoved vremena. Napišejo jih lahko računalniški ljubitelji ali zanesenjaki za lastno uporabo, poklicni programerji po naročilu uporabnikov, ali pa jih razvija stotine programerjev programske hiše. Glede na namen uporabe lahko programsko opremo razdelimo na:

- sistemsko programsko opremo – operacijski sistem
- razvojne in vzdrževalne programe in
- uporabniško ali aplikativno programsko opremo,

Sistemska programska oprema ali operacijski sistem (OS) so programi, ki so potrebni za delovanje in upravljanje računalnika in so stalno prisotni v računalniku. Uporabniku omogočajo enostavno in učinkovito uporabo računalnika in njegovih komponent. Nadzirajo izvajanje programov v določenem vrstnem redu, vršijo vhodno izhodne funkcije, dodeljujejo uporabo pomnilnika in nadzirajo pristop do računalnika ter do podatkov in programov na njem. Operacijski sistem je vmesnik med uporabnikom in računalnikom, zato določa uporabnost in zmogljivosti računalnika saj so od OS odvisni tudi enostavnost uporabe, varnost, učinkovitost in zanesljivost.

Razvojni in vzdrževalni programi so dodatni programi, ki za delovanje računalnika niso nujno potrebni. Omogočajo vzdrževanje in upravljanje s podatki in programi na računalniku ter omogočajo razvoj novih programov. Izdelujejo jih proizvajalci računalnikov ali programske hiše kot dodatek k operacijskemu sistemu. Ker so odvisni od strojne opreme, jih lahko uporabimo samo na določenem računalniku. Med te štejemo programe za pisanje in urejanje teksta, različna programska okolja, ipd.

Uporabniška ali aplikativna programska oprema so programi, namenjeni reševanju konkretnega problema. V novejšem času se pojavlja vse več splošnih programov, ki pokrivajo določeno področje uporabe, kot je oblikovanje besedil, delo s podatkovnimi bazami ali CAD. Te kot splošne programe izdelujejo programerske hiše za trg in so na voljo v izvedbah za različne tipe strojne opreme ali različne operacijske sisteme.

3.2 Operacijski sistem

Je skupek programov, ki so potrebni za delovanje in upravljanje računalnika in so stalno prisotni v računalniku. Nekoč so jih izdelovali proizvajalci računalnikov sami, dandanes je na voljo več različic operacijskih sistemov, predvsem za osebne računalnike in delovne postaje, le pri večjih sistemih so v rabi namenski OS.

Najbolj znani in razširjeni operacijski sistemi danes so: Windows, MacOS in Linux na osebnih računalnikih; ter Unix na delovnih postajah in strežnikih.

Običajno je OS na trajnem pomnilniku - disku, odtod tudi ime DOS (*angl. Disk Operating System*), v ROM je samo kratek del – npr. BIOS, ki ob zagonu računalnika naloži v pomnilnik potrebne dele OS. Ta način omogoča enostavne spremembe OS, zahteva pa del pomnilnika.

Sočasnost

Glede na število obdelav, ki jih lahko izvajamo na računalniku hkrati ločimo **enoopravilne** (*angl. single-tasking*) in **večopravilne** (*angl. multitasking*) operacijske sisteme. Pri večopravilnih na videz hkrati izvajamo več programov ali procesov, pišemo tekst s programom za oblikovanje besedil, izvajamo kalkulacije v tabelah in brsamo po spletu. Navidezno hkratnost doseže OS z

dodeljevanjem časovnih rezin (*angl. timesharing*). Procesor v kratkih časovnih korakih zaporedoma izvaja različne programe, tako da za uporabnika vsi napredujejo enako hitro. Izjema so več procesorski računalniki, kjer lahko vsak procesor izvaja svoj program. Večopravilni OS mora skrbeti tudi za deljenje zmogljivosti, pomnilnika, diskov, dodeljevanje zunanjih enot, ipd.

Dostopnost

Glede na število uporabnikov ločimo **enouporabniške** (*angl. single-user*) in **večuporabniške** (*angl. multi-user*) OS. Slednji omogočajo hkratno delo več uporabnikom na istem računalniku na ločenih vhodno/izhodnih enotah.

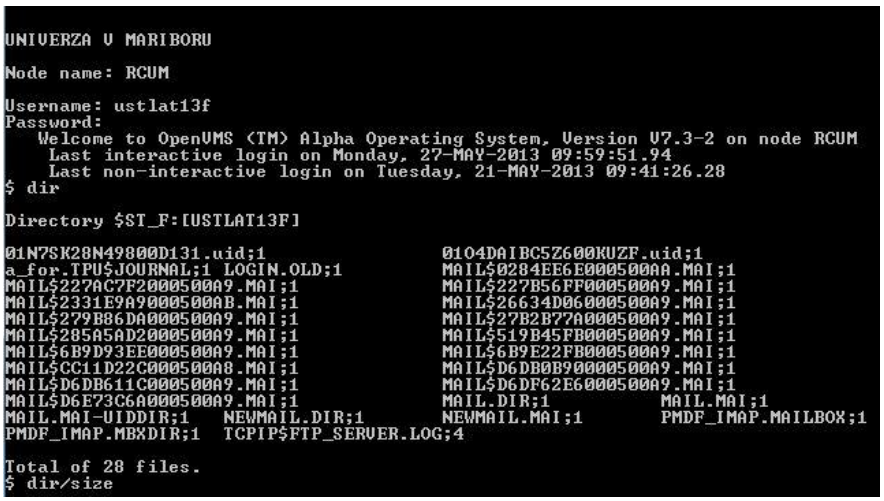
Pri več uporabniškem OS je zato potreben sistem zaščite, ki omogoča dostop do računalnika samo pooblaščenim uporabnikom in zagotavlja varnost podatkov vsakega uporabnika. To dosežemo z registracijo uporabnikov, kjer jim upravitelj sistema dodeli šifro in geslo. Vsak uporabnik se mora pred začetkom dela na računalniku prijaviti z veljavno šifro in geslom.

Enak sistem se je uveljavil tudi na enouporabniških sistemih, razlika je le ta, da lahko do računalnik dostopa le en uporabnik hkrati.

Enostavnost in učinkovitost

Neposredno delo z OS je potrebno samo pri razvoju novih programov in pri vzdrževanju računalnika. Zato mora biti delo z OS po eni strani enostavno, torej mora omogočati laičnemu uporabniku enostavno uporabo njegovih funkcij brez poglobljenega poznavanja OS, po drugi strani pa mora biti OS učinkovit, da lahko izkušeni uporabniki svoje naloge izvedejo hitro in zlahka. Žal si obe zahtevi nasprotujeta, tako da so nekateri OS učinkoviti in hitri kot je UNIX, vendar zahtevni za učenje, medtem ko so drugi enostavni kot Windows, a pri zahtevnejših opravilih neučinkoviti.

Pomemben element OS je **uporabniški vmesnik** (*angl. user interface*). Ta predstavlja povezavo med uporabnikom in operacijskim sistemom in v največji meri vpliva na enostavnost in hitrost uporabe. Uporabniku omogoča podajanje svojih ukazov in mu prikazuje rezultate dela. Po načinu dela ločimo **znakovne** in **grafične** uporabniške vmesnike (*angl. Graphical User Interface - GUI*).

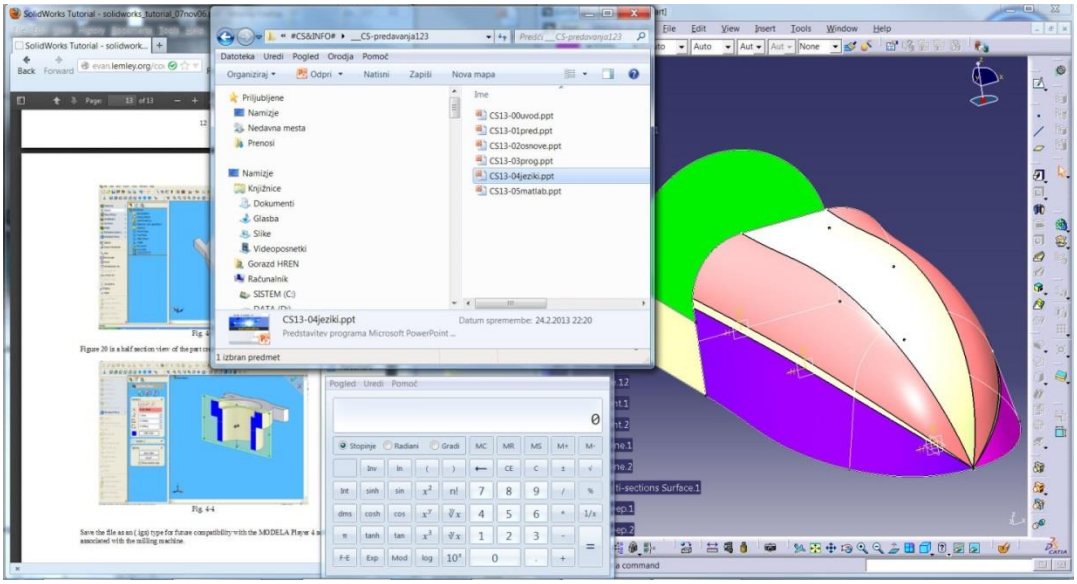


Slika 13: Primer znakovnega OS.

Znakovni uporabniški vmesnik potrebuje za delo samo alfanumerični - tekstovni zaslon in tipkovnico. Uporabnik vpisuje ukaze, zahteve in podatke, računalnik pa na zaslon izpisuje rezultate, obvestila in opozorila. Pri delu mora uporabnik poznati možne ukaze in obliko podajanja ukazov in podatkov.

Grafični uporabniški vmesnik potrebuje grafični zaslon in grafično vhodno napravo, kar je večinoma miška. GUI predstavlja delovno okolje, na kateri so grafično z ikonami prikazani

objekti kot so programi, podatkovne datoteke, diskovne in disketne enote in tiskalnik. Operacije simuliramo s premikanjem objektov s pomočjo vhodne naprave. Za delo s programom služijo delovna okna, ki nam omogočajo vpogled v dogajanje v programu. Hkrati lahko odpremo več oken in jih poljubno spreminjamo.



Slika 3: Primer grafičnega OS Windows.

3.3 Podatki in mediji

Računalnik je naprava za obdelavo podatkov. Te podatke mora računalnik med in po obdelavi shraniti na pomnilni medij, običajno disk. OS skrbi za organizacijo podatkov in njihovo dostopnost. Osnovna enota, ki jo obravnava OS je datoteka do katere posameznih zapisov OS nima direktnega dostopa, ampak upravlja le s celotno datoteko.

Tipi datotek

Podatki v datoteki so različnih tipov. Ločimo tekstovne in binarne datoteke.

Binarne datoteke vsebujejo podatke v obliki, ki jo razume računalnik. Tudi sami programi so datoteke, ki vsebujejo v binarni obliki zapisane ukaze procesorju. Poleg programov so binarne datoteke še podatki, ki jih lahko uporabi samo ustrezen program. Binarni zapis podatkov je kompaktnejši in omogoča programom hitrejše branje in zapisovanje. Ker je oblika binarnega zapisa odvisna od vrste računalnika in programa, ni možno direktno prenašanje podatkov v binarni obliki med različnimi tipi računalnikov ali različnimi programi. Zato se je uveljavilo nekaj standardnih zapisov binarnih podatkov, kot je GIF za slike, ki omogočajo prenose podatkov med različnimi programi in včasih tudi med različnimi tipi računalnikov.

Tekstovne datoteke so podatki zapisani s kodami za zapis znakov. Običajno uporabljamo ASCII kodno tabelo jim rečemo tudi ASCII datoteke. Njihovo vsebino lahko prikazujemo na zaslonu ali izpišemo na tiskalniku, s pomočjo urejevalnika tekstov pa lahko podatke vpisujemo ali popravljamo. Prav tako so zelo primerni za prenašanje podatkov med različnimi programi ali različnimi računalniki, saj je ASCII tabela standardna in jo poznajo vsi OS. Njihova slabost je predvsem v obsežnejšem zapisu in potrebi po pretvarjanju v ali iz binarne oblike, ki jo razumejo računalniki.

Označevanje datotek

Pri delu z datotekami mora biti vsaka datoteka enoznačno določena, sicer pride do dvoumnosti. Pravila za označevanje datotek so odvisna od operacijskega sistema vendar so osnovna načela označevanja pri vseh podobna.

Ime datoteke lahko uporabnik izbere poljubno, zaradi lažje uporabe naj izraža vsebino datoteke ali njen namen. Ime je lahko sestavljeno iz črk in številk. Nekaterih posebnih znakov kot so vejica, presledek, dvopičje, poševne črte, ipd. ne smemo uporabiti, saj imajo v dialogu z OS poseben pomen (odvisno sicer od OS).

Za lažje razpoznavanje vrste datotek uporabljamo pripone ali **tip** datoteke. Od imena ga ločimo s piko. Glede izbire znakov veljajo enaka pravila kot pri imenu. Tudi tip datoteke lahko uporabnik izbere poljubno, vendar je zaradi poenostavitve komunikacije z OS smiselno upoštevati priporočila. **docx** je standardna pripona za MSWord, dxf za AutoCAD, ipd.

Poleg imena datoteko opisujejo značilnosti:

velikost, ki pove koliko prostora datoteka zaseda na pomnilnem mediju,
datum nastanka, ki ga zapiše OS avtomatsko ob kreiranju datoteke,
datum zadnje spremembe ali uporabe,
lastnik datoteke, ki je pomemben na več uporabniških sistemih za uveljavljanje zaščite.

Organizacija datotek - imeniki

Današnji pomnilni mediji omogočajo shranjevanje vse večjega števila podatkov. Ker postane delo z velikim številom datotek nepregledno in iskanje zamudno uredimo datoteke v logične skupine. Vsaka skupina je prikazana kot datoteka – **imenik** ali **mapa** (*angl. folder*). To je binarna datoteka, ki vsebuje seznam vseh, datotek v skupini. Vsebine imenika ne moremo prikazati ali spreminjati neposredno, temveč le s pomočjo OS. Ker je vsak imenik tudi posebna oblika datoteke, seveda ni nobenih ovir za nadaljnje združevanje imenikov v skupine, oz. za vključevanje drugih imenikov - podimenikov v vsak imenik.

Struktura imenikov je hierarhična ali drevesna. Na vsakem pomnilnem mediju obstaja osnovni imenik (*angl.* root) in vsak imenik razen osnovnega ima svojega prednika in različno število naslednikov - lahko tudi nobenega. Vsaka datoteka v imeniku je dostopna samo preko svojih prednikov, zato moramo OS vselej podati zaporedje imenikov, ki nas pripelje do datoteke. To imenujemo tudi **pot** (*angl. path*).

3.4 Razvojni in vzdrževalni programi

Ti programi nisi nujno potrebni za delovanje računalnika a namenjeni vzdrževanju in nadzoru računalnika, varovanju pred virusi in prilagajanju delovnega okolja. Posebna skupina je namenjena programerjem za razvijanje novih programov. To so programska okolja s programski jeziki, ki so opisana v posebnem poglavju.

Programersko okolje

Je namenjeno izdelavi in razvijanju programov. Običajno vključuje programskemu jeziku prilagojen editor za pisanje programov, prevajalnik ali tolmač in orodja za testiranje programov in iskanje napak.

Orodja za delo s podatki

Za učinkovito in enostavno upravljanje datotek na računalniku večkrat OS ne zadošča. Zato uporabljamo dodatne pomožne programe za urejanje datotek, ki omogočajo:

nadzor in pregled podatkov, običajno z grafičnim prikazom drevesne strukture;
iskanje, premeščanje in kopiranje datotek;
zaščito datotek pred nepooblaščeno uporabo;
izdelava varnostnih kopij;
stiskanje podatkov zaradi prihranka pomnilnega prostora in lažje prenašanje
pregled in nadzor nad delovanjem pomnilnih medijev, odpravljanje napak in nastavitve;
zaščito računalnika pred virusi.

Novejši operacijski sistemi že vključujejo večino naštetih možnosti.

3.5 Uporabniški programi

Uporabniška ali aplikativna programska oprema so programi, namenjeni reševanju konkretnih problemov. Ker je razvoj novih programov zelo drag in ker je veliko nalog, ki jih rešujemo z računalnikom podobnih, se v novejšem času pojavlja vse več splošnih programov, ki pokrivajo določeno področje uporabe, npr. obdelavo besedil, delo z bazami podatkov ali inženirske naloge. Te programe običajno izdelujejo programske hiše za trg.

Urejanje in oblikovanje besedil

je verjetno najpogostejše opravilo na računalniku, saj je v zadnjih letih povsem izpodrinilo uporabo klasičnih pisalnih strojev. Programi za oblikovanje besedil (*angl. word processor*) so se razvili iz urejevalnikov tekstov njihov razvoj pa je bil odvisen od zmogljivosti tiskalnikov ter zmožnosti prikazovanja (enačb). Programi za urejanje in oblikovanje besedil temeljijo na načelu »kar vidiš to dobiš« (*angl. What You See Is What You Get - WYSIWYG*), kjer že med delom na slikovnem zaslonu vidimo končno podobo urejenega besedila. To je seveda omogočil hiter razvoj zmogljivosti računalnikov in grafičnih zaslonov, saj ta način dela zahteva zmogljivejši procesor ter predvsem grafični zaslon.

Najbolj znani in pri nas razširjeni programi te vrste so MSWord, Word Perfect in Write OO.o.

Elektronske preglednice

Elektronske preglednice (*angl. spreadsheet*) so se razširile kmalu po uveljavitvi osebnih računalnikov, saj so bile preglednice in tabelarični preračuni vsakodnevno opravilo. Ker je računalnik idealno orodje za tovrstne naloge, se je že leta 1979 na mikro računalniku Apple pojavila prva elektronska preglednica VisiCalc. V operacijskem sistemu Windows so danes najbolj znane aplikacije Microsoft Excel in Quattro Pro. Elektronske preglednice omogočajo avtomatizirane tabelarične preračune. Vrednosti v tabelah lahko tudi prikažemo v različnih vrstah grafikonov. Omogočajo tudi do podatkov v Internetu in objavljanje podatkov v spletu.

Predstavitve

Programi za predstavitev (*angl. presentation*) omogočajo izdelavo predstavitev, ki jih lahko izvajamo neposredno na računalniku običajno v povezavi s projektorjem. Ob tem lahko izkoristimo možnosti animacije, samodejnega izvajanja zaporednih tem ter vključevanja zvočnih in video posnetkov.

Podatkovne zbirke

so najpogosteje uporabljani programi na večjih računalniških sistemih, pa tudi na osebnih računalnikih so zelo pogosti. Podatkovna zbirka (*angl. database*) je urejena zbirka podatkov in program za dodajanje, urejanje, iskanje in dopolnjevanje podatkov. Osnovo predstavljajo **podatkovne tabele**, ki vsebujejo poljubno število zapisov (število je običajno omejeno z razpoložljivim pomnilnikom). Vsak zapis tvorijo podatkovna polja različnih tipov (datum, števila, valuto in drugo. Ker je vsak podatek predstavljen kot relacija med poljem in številko zapisa, jim pravimo tudi relacijske podatkovne zbirke.

Filtri ali poizvedbe (*angl. query*), omogočajo urejanje in izbiranje podatkov iz tabel. Poizvedbe omogočajo tudi povezovanje podatkov iz več različnih tabel ali več različnih podatkovnih zbirk.

Obrazci – »formularji« (*angl. forms*) so namenjeni delu s podatki. Temeljijo na tabelah ali poizvedbah, podatki so prikazani v podatkovnih poljih, kjer jih vpisujemo. Obrazci lahko vsebujejo tudi druge objekte: opise polj, ukazne gumbe, slike ipd. Vsakemu objektu lahko ob različnih dogodkih priredimo akcije, npr. tiskanje podatkov ob pritisku na gumb, ali urejanje po vnosu novega podatka.

Poročila (*angl. reports*) omogočajo urejeno prikazovanje in izpisovanje podatkov. Tudi poročila temeljijo na tabelah in poizvedbah, vsebujejo pa podatkovna polja, opise in slike. V podatkovna polja, lahko poleg podatkov vključimo tudi različne formule, npr. produkte posameznih polj, izračune delnih ali skupnih vsot in podobno.

Prva popularna podatkovna zbirka je bil program dBase, v okolju OS Windows so uveljavljeni programi Access, FoxPro in Paradox. Na večjih računalnikih je razširjen program Oracle.

Ukaze za delo s podatkovnimi zbirkami lahko uporabimo tudi za programiranje aplikacij. Razvili so se programski jeziki, ki izvajajo enake akcije kot podatkovne zbirke, Delphi, Visual Basic ipd.

Zbirke pisarniških orodij

Zaradi razširjenosti in uporabnosti večinoma zgoraj omenjene programe ponujajo kot zbirke orodij za pisarne. Le-te običajno vsebujejo program za oblikovanje besedil, elektronsko preglednico in predstavitve, običajno pa še program za risanje in elektronsko pošto.

Risanje

Programe, ki jih uporabljamo za risanje s pomočjo računalnika glede na način risanja in shranjevanja slik razvrstimo v dve osnovni skupini glede na predstavitev slike v računalniku: rastrske in vektorske.

Enostavnejši programi za rastrsko risanje so Slikar v Windows, zmogljivejši, namenjeni tudi obdelavi fotografij pa Corel PHOTO-PAINT, Adobe Photo Shop in podobni.

Potrebna količina pomnilnika za shranjevanje slike v točkovni obliki je zmnožek števila točk po dolžini, višina in logaritma števila barv. Tako slika velikosti 800x600 točk v 256 barvah zavzema 480Kb. Zato so razvili številne algoritme, ki takšno sliko komprimirajo, vendar je pri prenosu slik med različnimi grafičnimi programi potrebna pretvorba iz enega grafičnega formata v drugi. Za objavljanje slik v svetovnem spletu se uporabljata zlasti formata JPEG in GIF.

Pri programih za vektorsko risanje količina podatkov ni odvisna od velikosti risbe, marveč od količine in vrste elementov. Ker različni programi uporabljajo različne načine zapisa je pri prenosu risb med posameznimi programi potrebna pretvorba, obstajajo pa tudi standardi za pretvorbe, npr. IGES in STEP. Možna je tudi pretvorba v rastrski način zapisa slike. Na osebnih računalnikih je najbolj znan tovrstni program Corel Draw.

Zabava in izobraževanje

Z naraščanjem uporabe računalnikov in njihovo dostopnostjo predvsem mlajšim se hitro širi tržišče zabavnih in izobraževalnih programov, predvsem iger. Sodobne igre predstavljajo vrh razvoja računalniške grafike in zahtevajo velike računalniške zmogljivosti, predvsem pri grafiki, zvoku in uporabi spleta.

3.6 Preizkusni programi

Z razvojem omrežja, zlasti Interneta se je zelo razširila skupina preizkusnih programov (*angl. shareware*) pa tudi prostih (*angl. freeware*). Mnogi programerji, zlasti ljubiteljski, ki izdelajo programe zase, ponudijo te zainteresiranim uporabnikom v brezplačno uporabo ali pa v preizkušnjo za omejeno dobo, po tej dobi je potrebno program registrirati in plačati pristojbino. Nekatera podjetja uporabljajo tudi možnost "reklamnih" programov (*angl. addware*), kjer pri neregistrirani različici programa le-ta v okencu prikazuje različne oglase. Večinoma lahko brezplačne programe uporabljamo samo za privatno nekomercialno uporabo.

Med najbolj znanimi in razširjenimi brezplačnimi programi so OS Linux, pisarniški paket OpenOffice.org, spletna orodja iz zbirke Mozilla in drugi.

4 Osnove računalniških mrež in varnosti

4.1 Računalniške mreže

Internet je največje svetovno računalniško omrežje, ki združuje številna tehnološko heterogena lokalna omrežja v enoten prostor. Velikokrat ga zamenjujejo s posameznimi uslugami:WWW, e-mail, ipd.

Računalniško omrežje je dvojica ali skupina med seboj povezanih odvisnih ali neodvisnih računalniških sistemov, ki avtomatsko komunicirajo drug z drugim in si delijo vire (*angl. resources*) kot so podatki, programska in strojna oprema.

Računalniška omrežja so danes najhitreje razvijajoča se veja računalništva, ki zajema tudi telekomunikacije. Beseda telekomunikacije je sestavljena iz grške predpone *tele*=daleč in latinske besede *communicare*=deliti. Namen uporabe računalniških omrežij je:

- komunikacija med uporabniki in/ali programi na različnih računalnikih,
    - delitev skupnih perifernih enot med računalniki, npr. tiskalnikov,diskov itd. in s tem,
    - boljši izkoristek vhodno/izhodnih enot,
- učinkovitejše vzdrževanje programov in podatkov,
- povečanje zanesljivosti zahtevnih računalniških sistemov.

Povezovanje računalnikov je večplastno. Na najvišjem nivoju so storitve, ki jih povezovanje računalnikov nudi uporabniku. Gre za programe – aplikacije, ki omogočajo komuniciranje dveh uporabnikov oz. uporabo virov oddaljenega računalnika.

Vmesna plast je transportna - zagotavlja pravilno usmerjanje podatkov od izvora do ponora. Na najnižjem nivoju pa so komunikacije – prenosni medij in ustrezni vmesniki, ki omogočajo podatke iz enega računalnika prenesti do drugega. Vsaka plast ima tudi dogovorjena pravila – protokole, ki omogočajo povezovanje in prenos podatkov med dvema računalnikoma.

4.1.1 Storitve

Osnovne storitve, ki jih uporabniku nudi uporaba omrežja so:

na relaciji uporabnik - računalnik

- Uporaba podatkov na drugem računalniku
- Uporaba virov drugega računalnika npr. tiskalnik, disk, procesor
- Delo na oddaljenem računalniku – navidezni terminal, telnet
- Prenos datotek

Na relaciji uporabnik – uporabnik

- Elektronska pošta – pošiljanje sporočil. Vsebuje lahko tekst, sliko in zvok.
- Konference – news – sodelovanje v razpravi o določeni temi.
- Neposredna komunikacija – klepet (angl. chat)

4.1.2 Obseg omrežja

Glede na krajevno razprostranost omrežja jih delimo v dve skupini. Te so se oblikovale z razvojem tehnologije, tako da za vsako od njih običajno uporabljamo drugačno tehnologijo.

Lokalno omrežje (angl. Local Area Network – LAN)

Na manjšem prostoru – običajno v eni sobi ali zgradbi ali organizaciji.

Globalno omrežje (angl. Wide Area Network – WAN)

Povezuje lokalna omrežja v globalno. Uporabljajo se povezave velikih hitrosti, tudi satelitski prenosi.

4.1.3 Komunikacijski medij

Za prenos signalov med enotami lahko uporabimo:

Žični par. Običajno uporabimo izolirane neoklopljene žice, ki jim lastnosti izboljšamo z zvijanjem okoli vzdolžne osi.
Koaksialni kabel, kjer je sredica oklopljena zaradi preprečevanja vpliva motenj.
Optični kabel, ki omogoča hitre prenose na daljših razdaljah.
Infrardeči prenosi na krajših razdaljah brez ovir, npr. povezava prenosnega računalnika s stacionarnim.
Zemeljski radijski prenosi, kjer ni možna fizična povezava.
Satelitski prenosi za globalno povezovanje.

Telefonsko omrežje

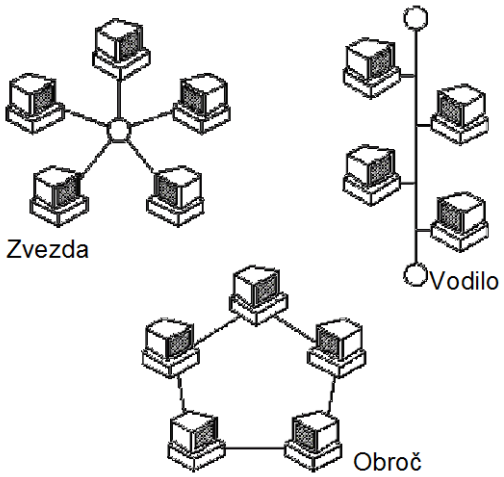
Za povezovanje predvsem na daljše razdalje velikokrat uporabimo že obstoječa komunikacijska omrežja, npr. telefonsko ali električno, v novejšem času pa tudi kabelsko TV. Najpogosteje uporabimo kabelska omrežja, ki omogočajo prenos analognih ali digitalnih signalov. Uporabimo lahko najete linije – direktna povezava med vstopno in končno točko, ali komutirane linije. Pri uporabi analognih linij potrebujemo še napravo, ki pretvarja signale iz analogne v digitalno obliko in nazaj, pri tem pa izvaja še funkcije potrebne za vzpostavljanje zveze – izbira klicnih številk in odziv na klic. To opravlja klicni modem.

4.1.4 Oblike povezav – topologija

Glede na način povezave posameznih računalnikov v omrežju poznamo različne topologije.

Zvezda

Pri zvezdni topologiji imamo centralno vozlišče – koncentrator (*angl. hub*), na katerega so priključene vse enote. Prednost je v enostavni povezavi in usmerjanju prometa, slabosti pa v dolžini povezav, občutljivosti na izpad vozlišča – omrežje takrat ne deluje in v obremenitvi vozlišča pri večjem prometu. Pri prekinitvi ene povezave se prekine samo zveza ene enote z vozliščem. Več vozlišč lahko povežemo v drevesno topologijo ali v obroč.



Slika 4: Topologije omrežja

Vodilo

Pri tej topologiji so vse enote priključene na osrednji kabel, ki mora biti na obeh koncih zaključen s posebnimi končniki – terminatorji. Prednosti so v enostavnosti in cenenosti. Dodajanje novih enot je enostavno. Pri večjem številu priključenih enot in gostejšem prometu prepustnost zelo upade. Prav tako je zelo težaven nadzor nad omrežjem. Pri prekinitvi vodila promet po omrežju ni možen.

Obroč

Vse naprave so povezane ena z drugo v obliki sklenjenega obroča, tako da je vsaka povezana z dvema sosednjima. Ta topologija je sorazmerno draga, vendar omogoča hitre prenose in daljše razdalje, saj vsaka naprava deluje tudi kot ojačevalec.

4.1.5 Arhitektura omrežja

Po načinu uporabe omrežnih storitev ločimo pristopa:

Vsak z vsakim *(angl. peer-to-peer)*

kjer so vsi sistemi enakovredni in komunicirajo drug z drugim. Ta oblika omrežja je na splošno enostavnejša in cenejša, vendar pri velikih obremenitvah manj učinkovita. Vsak uporabnik lahko vire svojega računalnika – datoteke, tiskalnike - da v skupno rabo (*angl. sharing*), do teh virov pa dostopajo uporabniki drugih računalnikov.

Odjemalec/strežnik *(angl. client/server)*

Sistem odjemalec/strežnik sestavlja dvojica računalnikov. Prvi, s katerim dela uporabnik, izvaja program – odjemalec, ki pošilja zahteve strežniku. Ta - običajno zmogljivejši računalnik – jim streže, pri tem pa uporablja:

- informacijsko skladišče s podatki, programi itd.
- sistem – program za posredovanje podatkov, ki jih zahteva odjemalec.

Na tej zasnovi temeljijo tudi osnovne storite Interneta - elektronska pošta, novice, svetovni splet.

4.1.6 Povezovanje omrežij

Skrajna dolžina prenosa podatkov med dvema točkama je omejena z izbiro tehnologije in prenosnega medija. Kadar želimo podatke pošiljati na večji razdalji, moramo signale v vodniku ojačiti.

Ponavljalnik *(angl. repeater)*

Ojači bitne signale, ki slabijo vzdolž prenosnega medija. Kadar povežemo dve veji omrežja na različnem mediju, skrbi tudi za pretvorbo signala, npr. med bakrenim in optičnim. Pri topologiji obroča vsaka postaja deluje tudi kot ponavljalnik.

Most *(angl. bridge)*

Uporabljamo za povezovanje različnih lokalnih omrežij. Njegova naloga je pretvarjanje podatkov med različnimi protokoli. Izboljša prepustnost omrežja, saj promet znotraj ene veje ne obremenjuje drugih vej omrežja.

Usmerjevalnik *(angl. router)*

Omogoča povezovanje različnih tipov omrežij in skrbijo za pravilno usmerjanje podatkov od pošiljatelja do naslovnika.

4.1.7 Brezžična omrežja

Brezžično lokalno omrežje (angl. Wireless LAN – WLAN) je običajno izvedeno po standardu IEEE 802.11b ali 802.11g. Prvi omogoča največje prenose do 11Mb/s, slednji pa 54Mb/s. Obrežje za prenos uporablja frekvenco 2,4GHz, domet pa je do 300m na odprtem in nekaj 10m v zaprtih prostorih, odvisno od konstrukcije.

Uporablja se predvsem za povezovanje prenosnih naprav v lokalno omrežje in dostop do Interneta. Za povezavo potrebujemo dostopno točko (*angl. Access Point*), ki jo povežemo v lokalno omrežje, v računalniku pa brezžično omrežno kartico z anteno – v nekaterih je že. Brezžilne omrežne kartice so lahko izdelane tudi kot priključki na vodilu USB. Dostopne točke so pogosto vgrajene v brezžični usmerjevalnik (*Wireless Router*), ki omogoča poleg brezžične še ožičeno povezavo.

4.2 Računalniška varnost in etika

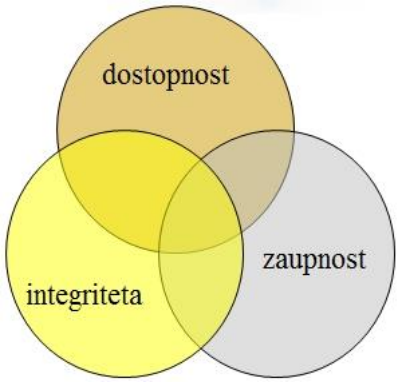
Klasifikacija nevarnosti

Osnovni deli računalniškega sistema so: strojna oprema, programska oprema in podatki. Ločimo štiri tipe nevarnosti:

- **Prekinitev** – element sistema je izgubljen, nedostopen ali neuporaben (npr. uničenje strojne opreme, odstranitev programa ali podatka, ipd.)
- **Prestrezanje** - kadar nepooblaščena stranka (lahko je oseba, program ali drug računalniški sistem) pridobi dostop do elementa računalniškega sistema (kopiranje programa ali podatkov, prisluškovanje na omrežju, ipd.). Prestrezanje je izjemno težko ugotoviti ali dokazati (kadar gre zgolj za prestrezanje), saj ne pušča nikakršnih sledi.
- **Spreminjanje** - kadar ne gre zgolj za dostop do elementa računalniškega sistema, temveč tudi njegovo spreminjanje (spremenijo se lahko vrednosti podatkov, delovanje programa, strojna oprema, ipd.). Nekatere spremembe lahko preprosto odkrijemo, druge bolj premišljene je težko ali nemogoče odkriti.
- **Ponarejanje** - kadar se naredi nekaj novega, kar pa ni ustvarila pooblaščena oseba (npr. vstavljanje novih zapisov, ponarejanje programov, podpisov ipd.)

Karakteristike zagotavljanja varnosti

- **Zaupnost** - pomeni, da lahko do določenega elementa računalniškega sistema dostopa samo za to pooblaščen (avtoriziran) uporabnik (človek, računalniški sistem, ipd.). Dostop pomeni branje v kakršnikoli obliki (kopiranje, pregled, tiskanje ali samo vedenje, da obstaja).
- **Integriteta** - pomeni, da lahko določen element računalniškega sistema spreminja samo za to pooblaščen uporabnik in to na predviden način. Pod spreminjanje uvrščamo pisanje, spreminjanje, spreminjanje statusa, brisanje in ustvarjanje.
- **Dostopnost** - pomeni, da je element računalniškega sistema v vsakem trenutku na voljo pooblaščenemu uporabniku.



Slika 14: Karakteristike zagotavljanja varnosti.

Vse tri karakteristike skupaj zagotavljajo varnost računalniškega sistema na uporabniku koristen način. Če namreč ne zadostimo vsem trem zahtevam, je sistem neuporaben. Lahko bi zagotovili popolno zaupnost podatkov in preprečili dostop do njih komerkoli, vendar bi v takšnem primeru podatki bili neuporabni (kršimo karakteristiko dostopnosti).

Vse tri karakteristike se seveda medsebojno prekrivajo kot prikazuje slika, namen varnosti pa je zagotoviti vse tri, torej presek.

Računalnik ima več virov ranljivosti:

- Strojna oprema:
  - splošno varovanje: izolacija, alarmni sistemi za vročino, dim, vlom; naprave za varstvo pred požari, kartice za verifikacijo vstopa.

    - fizično varovanje: backup, UPS-neprekinjeno ele.napajanje, RAID5, kontrola dostopa, ključi?
- Programska oprema:
    - spremembe programov -1cent s vsakega računa na svoj račun – preverjanje sprememb.
    - Virusi – antivirusni programi.
    - Podatkovna komunikacija –rezanje kablov dostop do brezžičnega prenosa– kriptografija.
- Podatkovne baze.
- Podatkovna komunikacija (tudi Internet).
- Uporabniki.

Varnost dostopa je mogoče zagotoviti z uporabo gesel (večkratne spremembe) ali z biometričnimi identifikacijski sistemi: prsni odtis, obrazni termogrami, razpoznava obraza, ušes, očesne šarenice, žil v roki, skratka vseh biometričnih lastnosti, ki so sorazmerno hitro dosegljive in delajo človeka unikatnega.

Računalniški virus je programček, ki
- se naloži na disk brez naše privolitve in se pripne na datoteko,
- miruje, dokler ne storimo kakšne akcije, kot:
    - vklopimo računalnik,
    - poženemo drug program
- na skrivaj pomnoži samega sebe na druge dele diska ali na drugem računalniku.
- lahko samo prikaže sporočilo. V skrajnem primeru zbriše diske ali poškoduje strojno opremo!
- Ločimo:
    - logične viruse (se sprožijo ob določeni akciji)
    - časovne viruse (se sprožijo na določeni datum)

Računalniški črv je podvrsta virusa in deluje samodejno. Deluje med omrežji in spremeni ali prepiše podatke
- je podoben virusu, vendar uporablja omrežne povezave za pomik od računalnika do računalnika
- nato deluje enako, kot običajni virus, samo s to razliko, da se ne pripne na datoteko
- večinoma izvajajo DOS (Denial of Service). Predstavlja tip napada na omrežje, ki prekine dotok informacij tako, da napolni določeno omrežje z nepomembnim in ogromnim prometom.
- pogosto prevzamejo tudi kontrolo nad računalnikom

Trojanski konj je poseben tip računalniškega programa, ki deluje v okviru običajno znane aplikacije, vendar vsebuje nezaželene in razdiralne akcije
- nahajajo se v priljubljenih programih, ki jih dobimo drugje kot na originalni varni lokaciji
- nato delujejo enako kot običajni virusi, samo s to razliko, da se ne pomnožujejo
- primer: programček za odstranitev virusov v sebi vključuje viruse!

Piškotki so sporočila, ki jih pošilja spletni strežnik spletnemu čitalcu. Običajno:
- zbirajo demografske informacije, kdo bere spletne strani
- lažje poosebljajo spletne strani posameznega uporabnika (primer: Amazon)
- se uporabljajo za spremljavo reklam na spletni strani
- piškotki pa se lahko uporabijo tudi za:
    - zbiranje informacij o uporabniku za potrebe napada z nezaželjenimi oglasi in maili,
    - zbiranje informacij o geslih,
    - vendar ne vsebujejo virusov.

Preusmerjanje:
- klic določene spletne strani lahko vodi k temu, da pridemo v resnici na drug računalnik,
- primer: cnn.com -> 164.8.6.100 <> napad -> 298.8.9.56,
- informacije na našem računalniku se preberejo in po potrebi spremenijo.

Vohunjenje:
- delujejo v ozadju in snemajo celotno dogajanje (tipkanje, stanje na zaslonu, pisanje mail-ov) brez vednosti uporabnika,
- delujejo v ozadju in preverjajo pošiljanje podatkov v Internet brez vednosti uporabnika.

Najboljša zaščita računalnika je preventiva, pri kateri je najbolj znano pravilo:
NNN – ne odpri, ne kupi, ne odgovori !!!
- ne nalagajmo datotek iz neznanih lokacij,
- nabavimo si dober antivirusni program,
- ne odpirajmo e-mail priponk neznanih oseb,
- zbrišimo takoj e-mail od neznanih oseb,
- poženimo antivirusni program vsaj vsak teden,
- postavimo si požarni zid.

Kdo so napadalci, ki ogrožajo računalnike? Na kratko lahko Ljudi, ki namerno ogrožajo varnost računalniških sistemov, delimo v tri osnovne skupine:
- **Amaterji** so ljudje, ki jih zamika priložnost in neorganizirano in nenačrtovano poskušajo ogroziti varnost računalniškega sistema. Včasih se niti ne zavedajo veličine svojega dejanja, dokler jih ne ujamejo. Večino ugotovljenih vdorov so zagrešili prav amaterji.
- **Hackerji** so ljudje, ki praviloma iz užitka in za lastno veselje vdirajo ali poškodujejo računalniški sistem. Njihovo početje se strogo kaznuje in je z zakonom prepovedano.
- **Profesionalni vlomilci** so ljudje, ki svoje kriminalno početje vse bolj selijo v računalniške vode. Gre za profesionalne vlomilce in kriminalce, ki poskušajo na ta način obogateti.

Metode obrambe so, kot je običajno, en korak za napadalci, pa vendar veljajo določena pravila in postopki, ki lahko nevarnost napada vsaj zmanjšajo:
- enkripcija; zanesljiva oblika zaščite podatkov je kodiranje le teh. Kodiranje je preoblikovanje podatkov v za človeka brezpomensko zaporedje znakov. Na ta način je prestrezanje, spreminjanje in ponarejanje podatkov brezpredmetno. Edini način ogrožanja varnosti je uničenje, kar pa je, kot smo že omenili, enostavno ugotoviti, po drugi strani pa enostavneje varovati (rezervne kopije, replikacije).
- programska oprema; lahko poskrbi za velik del varnosti v računalniškem sistemu. Nadzor, ki ga vrši programska oprema, lahko razdelimo na tri nivoje:
  - interna kontrola (deli programa nadzorujejo dovoljenja za opravljanje operacij; v to skupino sodi varnost, o kateri govorimo pri programskem jeziku Java).
  - nadzor operacijskega sistema; naloga OS je, da izvaja preverjanje uporabnikov in jim dodeljuje pravice za opravljanje operacij skladno z njihovim statusom.
  - nadzor razvoja programske opreme; vse se začne pri zasnovi programske opreme, če se že tukaj omogočijo varnostne luknje, potem je varovanje sistema zelo oteženo.
- strojna oprema namenjena zaščiti,
- politika in določila,
- fizična zaščita.

Kodiranje ali enkripcija

Kodiranje sporočil je znano iz zgodovine človeštva, največji razvoj pa je doseglo v času druge svetovne vojne. Kodiranje lahko razdelimo v tri skupine:

- simetrično kodiranje
- kodiranje s privatnim in javnim ključem
- digitalno podpisovanje

Enkripcija podatkov je preoblikovanje zaporedja bitov v drugačno, človeku brezpomensko zaporedje.
Zaradi razumevanja nadaljnje vsebine bomo na kratko razložili princip uporabe enega oziroma dveh ključev.

Uporaba simetričnega ključa:

Je enostavnejši in hitrejši od principa z dvema ključema in se uporablja za kodiranje podatkov pred pošiljanjem. Za kodiranje in dekodiranje se uporablja isti ključ. Problem je prenos ključa.

Uporaba dveh ključev:

Gre za znan princip uporabe javnega in privatnega (*public/private key encryption*) para ključev. Če želi oseba A poslati podatke osebi B, potrebuje javni ključ osebe B. Z njim zakodira podatke in jih pošlje osebi B. Oseba B s svojim privatnim ključem dekodira podatke. Prednost tega načina je brezskrbno prenašanje ključa za kodiranje, saj nam le ta pri dekodiranju ne pomaga. Prav tako je iz njega praktično nemogoče določiti ustrezen privatni ključ.

Simetrično kodiranje

- uporabljaja samo en ključ,
- ključ mora ostati tajen,
- algoritmi so enostavnejši in hitrejši,
- uporabljajo se za kodiranje velike količine podatkov.

Pri sistemih s simetričnim ključem uporabljamo samo en ključ tako za kodiranje, kot tudi dekodiranje podatkov. Ključ mora ostati tajen. Algoritmi so praviloma hitrejši, zato se uporabljajo za kodiranje večje količine podatkov. Sistem zagotavlja tudi pristnost (*authentication*) pošiljatelja, saj lahko le sporočilo, ki je bilo kodirano s pošiljateljevim ključem dekodiramo z našim ključem (ki je enak). Problem se pojavi pri distribuciji ključa, saj ga moramo dostaviti na varen način. Ključ nam namreč omogoča vpogled v vsa zakodirana sporočila kot tudi spremembo in ponarejanje sporočil. Prav distribucija ključev je lahko pri tovrstnem kodiranju največji problem. Ena izmed rešitev je, da ključ pošljemo po delih preko različnih distribucijskih kanalov kot to prikazuje slika.

Rivest-Shamir-Adelman (RSA) algoritem za kodiranje je bil predstavljen leta 1978 in temelji na zahtevnih matematičnih problemih. Kljub podvrženosti obširnim kriptografskim analizam do danes ni bil razbit oziroma niso bile odkrite pomanjkljivosti. RSA povezuje teorijo velikih števil z zahtevnostjo iskanja nedeljivih števil in celoštevilskim deljenjem.

Potek algoritma je naslednji. Za kodiranje in dekodiranje uporablja dva ključa imenujmo **d** in **e**, ki sta pravzaprav zamenljiva. Besedilo (**P**) se zakodira kot $\mathbf{P}^{e}$ mod **n**. Celoštevilčno deljenje oteži pridobivanje osnovne tekstovne vrednosti P. Da bi dobili P potrebujemo pazljivo izbrano število **d**, ki ga imenujemo ključ. Z njim opravimo naslednji izračun $(P^{e})^{d}$ mod n = P, da pridobimo originalno besedilo. Kodiranje temelji na množenju velikih števil zato izračun brez poznavanja **d** praktično ni izvedljiv, saj je časovna zahtevnost najhitrejšega algoritma eksponentna.

Digitalni podpis

Digitalni podpis temelji na enkripciji podatkov s pomočjo ključa. Ključ je podatek, ki ga uporabite pri pretvorbi originalnih podatkov v obliko, ki je nerazumljiva (kodirana, šifrirana). Če želite ponovno originalno obliko podatkov potrebujete isti ali nek drug ustrezen (ne katerikoli) ključ. Z njim podatke dekodirate. V ta namen je razvitih več postopkov. Digitalni podpis temelji na postopku uporabe javnega ključa (public key encryption).

Postopek: uporabnik si ustvari dva ključa. Prvi se imenuje javni, drugi pa privatni. Privatni ključ je ključ, ki dekodira podatke kodirane z javnim ključem. Javni ključ je dobil ime na podlagi njegove uporabe. Uporabnik namreč le tega javno objavi (prenese ga na ustrezen temu namenjen strežnik) in s tem omogoči vsem, ki mu želijo poslati podatke njegovo uporabo. Podatke zakodirane z javnim ključem odklene (odkodira) privatni ključ. Z javnim ključem lahko podatke samo zakodiramo. Prav tako je iz javnega ključa praktično nemogoče dobiti privatni ključ.

Tehniko javnega ključa lahko uporabimo za podpisovanje podatkov. Postopek je naslednji: Z uporabo privatnega ključa izdelamo digitalni podpis, ki temelji na določenih podatkih. Uporabnik, ki želi preveriti verodostojnost teh podatkov potrebuje originalni podatke, digitalni podpis, ki temelji na teh podatkih in javni ključ uporabljenega privatnega ključa. Z uporabo javnega ključa lahko preverimo ali se podpis in podatki ujemajo.

Z uporabo enkripcije lahko na podlagi nekih podatkov (besedilo, aplet, ipd.) izdelamo tem podatkom lastno različico podatkov, ki je količinsko veliko manjša in jo imenujemo izvleček (*angl. digest*). Izvleček ponavadi imenujemo kar digitalni podpis. Digitalni podpis ne predstavlja celote prvotnih podatkov zato je iz njega nemogoče pridobiti originalne podatke. Gre za kodiranje le v eno smer. Originalnih podatkov ne moremo dobiti, lahko pa, če originalne podatke imamo preverimo njihovo pristnost oziroma ali so bili podpisani z določenim ključem. Jasno je, da ob kakršni koli spremembi originalnih podatkov digitalni podpis ne bo več enak.

Varnostna priporočila

- imejte dva mail naslova:
  - privatni mail naslov za prijatelje
  - zaščitni mail naslov za spletne strani (hotmail, yahoo, mail.si ...)
- nikoli ne vpisujte številk bančnih kartic
- nikoli ne dovolite shranjevanje imen in gesel na računalnik
- ne vpisujte osebnih podatkov, če to le ni res potrebno!
- uporabljajte posodobljene antivirusne programe: AVG: http://www.grisoft.com, Norton Antivirus, F-Prot, McAffee
- uporabljajte posodobljene odstranjevalnike vohunskih programov: Ad-aware (http://www.lavasoftusa.com/); AntiSpyware (http://www.microsoft.com/)
- definirajte požarni zid.

5 PROGRAMSKI JEZIKI

5.1 Uvod

Znanje programiranja je zelo dobrodošlo na številnih področjih v vsakdanjem življenju, saj dandanes uporabljamo veliko naprav, ki so programirane. Poznavanje, kako nek sistem deluje (tudi bankomat) s tem avtomatsko pomeni boljšo uporabo teh sistemov in hkrati znanje kaj narediti, če pri uporabi ne dobimo želenega rezultata. Največ kar pridobimo od učenja programiranja je zagotovo **logično algoritmično razmišljanje**, ki danes sodi v funkcionalno znanje vsakega posameznika, četudi ne uporablja računalnikov (če je to še sploh mogoče). Inženirji tehnike zagotovo niso tisti, ki bi se lahko uporabi računalnika odpovedali, saj so danes že odvisni od učinkovite uporabe le-teh in uporaba računalnika sodi v osnovno funkcionalno pismenost. Logično algoritmično razmišljanje pomeni tudi branje navodil za uporabo naprav, uporabljanje literature predvsem elektronskih priročnikov za programje. Namen tega poglavja ni specialistično znanje programiranja v nekem programskem jeziku, temveč vzpodbuditi in razviti logično algoritmično razmišljanje!

Namen poglavja je naučiti študente kako analizirati in programirati probleme za reševanje z računalnikom, sestaviti algoritme in diagrame poteka in pretvoriti algoritme s kodiranjem v delujoče računalniške programe. Ker je za kodiranje programov potrebnih kar nekaj izkušenj in znanja, je poudarek na analizi problema in kreiranju algoritma preden se lotimo kodiranja. Kodiranje se izvaja v programskem jeziku, ki je primeren glede na tip problema.

Različni uporabniški programi so zapisani za najširšo populacijo uporabnikov in čeprav vsebujejo veliko ukazov in nastavitev, se velikokrat izkaže, da potrebujemo nekaj bolj specialnega, kar avtorji niso predvideli ali pa ni dovolj splošno in zanimivo za večino uporabnikov. V tem primeru imamo možnost, da uporabnik sam sestavi potrebne ukaze v programsko kodo. To je zapis navodil v določenem programskem jeziku znotraj programskega orodja, ki ne zahteva posebnih postopkov, saj vključevanje kode izvede programsko okolje. To so v večini makro jeziki, ki jih srečamo v vseh Microsoftovih aplikacijah, CAD (večinoma VisualBasic for Applications)… Makro programi se velikokrat izkažejo zelo uporabni tudi na nivoju operacijskega sistema, kadar gre za večkratno ponavljanje istega zaporednega nabora ukazov.

Poznavanje programiranja je pri inženirskem delu prej nujnost kot dodatno znanje ali veščina. Nekateri osnovni razlogi zakaj je dobro poznavanje programiranja so med drugimi naslednji:

> *Bistveno boljša uporaba obstoječih programskih orodij, tudi specializiranih, kot so programi za pregled tabel, modeliranja v prostoru ali programov za kalkulacije.*
>
> *Lažje branje navodil in postopkov, ki so večinoma zapisani v obliki algoritmov.*
>
> *Razumno iskanje in naročanje programskih orodij (predstavitev problemov).*
>
> *Vsak inženir je »naprednejši« računalniški uporabnik in se sreča s programiranjem pri vsakdanjem delu; makro ukazi, priprava spletnih strani, popravljanje in dodajanje programov, kalkulacije, pomoč več uporabnikom …*

Za začetek nekaj terminologije, ki se v tem poglavju uporablja:

Računalnik – je stroj, ki je sposoben shranjevanja in izvajanja niza inštrukcij, ki jih imenujemo programi, z namenom reševanja določenega problema.

Platforma – je kombinacija strojne opreme in operacijskega sistema.

Inštrukcija – enostavno izvedljiv ukaz na računalniku (seštevanje dveh števil, prenos v pomnilnik).

Strojna koda – je edini programski jezik, ki ga računalnik »razume« in ga je sposoben izvajati; sestavljata ga binarna znaka 0 in 1.

Programski jezik – je zbirka pravil (slovnica, sintaksa) za kodiranje inštrukcij za računalnik. Ima svojo slovnico »sintakso« za programiranje.

Izvorna koda – je večinoma zapisana v enem od višjenivojskih programskih jezikov, lahko jo bere človek, potrebno jo je prevesti v strojno kodo (binarno) s prevajanjem ali tolmačenjem.

Nižje-nivojski programski jezik – je odvisen od strojne opreme (zbirni jezik) in izvaja inštrukcije neposredno v procesorju.

Višje-nivojski programski jezik – uporaba simbolnega zapisa, bere ga lahko uporabnik (Fortran, C++, Pascal, Java, Matlab, Scilab), potrebno pa ga je pretvoriti v strojno kodo za izvajanje.

5.2 Program, programski jeziki in programsko okolje

Programske jezike potrebujemo kot vmesnik med človekom in računalnikom, torej možnost komunikacije med človekom in računalnikom. Računalnik uporablja *strojni jezik*. Kodiranje v strojnem jeziku je izredno nepregledno in zapleteno, dodatna težava je tudi v tem, da ima vsak procesor drugačen strojni jezik. Uporaba strojnega jezika se je z razvojem poskušala poenostaviti s kraticami (na primer: ADD za seštevanje), s čemer je nastal *zbirni jezik* (*ang. asembler*). Ta način programiranja je zahteval postopek, ki je pretvoril zbirni jezik v strojni, ki ga računalnik lahko izvaja. Ta postopek, ki je kratice prevedel v strojno kodo, je bil programiran in imenovan prevajalnik. Zbirni jezik je bil le skromna izboljšava pri programiranju. Nadaljnji razvoj je prinesel prenos programskih jezikov na višjo, simbolno raven. Razvili so se zelo različni programski jeziki, odvisni predvsem od namena uporabe (numerični, poslovni, …), ki pa imajo skupno lastnost: ukazi označujejo niz opravil, ki se pri prevajanju prevedejo v več instrukcij zbirnega jezika in jih zato imenujemo višji programski jeziki.

Potrebujemo torej skupni jezik, ki omogoča avtomatski prevod v strojni jezik na višji simbolni ravni. Ker potrebujemo enak jezik za vse računalnike, to zagotovo ne more biti pogovorni jezik, ki jih je zelo veliko, so slovnično raznoliki in kompleksni in ne omogočajo avtomatskega prevajanja v strojni jezik. Uporabljamo programske jezike, ki so dovolj splošni za človeško razumevanje in dovolj natančni za avtomatsko prevajanje v strojni jezik.

Besedilo programa imenujemo programska koda ali izvorni program (*ang. source*), ki ga še ni mogoče izvajati. Pred zagonom ga je potrebno prevesti v strojni jezik, kar naredimo s prevajalnikom ali tolmačem. Prevajalnik in tolmač sta programa, ki izvorno kodo uredita v strojno obliko. Razlika med prevajanjem in tolmačenjem je nazorno prikazana v nadaljevanju. Izvorna koda ima za imenom pripono, ki nakazuje v katerem programskem jeziku je koda zapisana. Program, ki ga lahko izvajamo na računalniku se imenuje izvršni program. Tako imamo za javanske programe izvorno kodo zapisano v obliki *ime*.java, izvršna koda ima končnico .class.

Značilnost višjih programskih jezikov je tudi ta, da so vključeni v dodatna orodja oziroma okolje za programiranje. Torej, k nekemu programskemu jeziku sodi urejevalnik za zapis kode, prevajalnik, program za iskanje napak razhroščevalnik (ang. *debuger*), povezovalnik različnih programskih enot, kar vse skupaj imenujemo programsko okolje. Programsko okolje je podobno urejevalniku besedila, ki vključuje odpiranje in shranjevanje datotek in podobno, dodatne možnosti pa so pri odpiranju projektov, zaznavanje programskih besed, preverjanje zaključenosti blokov ukazov in podobno, pa seveda tudi možnosti prevajanja, in testiranja programov. Vsi novejši jeziki imajo tudi svoja urejena programerska okolja, imamo pa tudi urejevalnike besedil in okolja, ki podpirajo kodiranje v več programskih jezikih.

Programskih jezikov je pravzaprav izredno veliko in se razlikujejo predvsem po tem čemu so namenjeni. Zato jih je izredno težko razvrščati po lastnostih. Fortran velja za enostavnega pri

reševanju matematičnih problemov, pa okornega pri delu z izpisom na zaslon v okolju Windows, za VisualBasic pa velja ravno obratno.

Programski jezik je sistem simbolov za opis izvajanja aktivnosti v obliki, ki ga razumeta tako računalnik kot človek, pri čemer z izvajanjem mislimo na vse aktivnosti, ki jih zmore računalnik. Programski jezik je torej vmesnik med človekom in strojem. Da je programski jezik uporaben, mora biti vmesnik za vso strojno opremo (*hardware*) in programje (*software*), ki jih uporabljamo. Prav zaradi naštetega mora imeti programski jezik natančno definirano slovnico - *sintakso*, da lahko stroj (računalnik) bere in razčleni program in natančno definirano pomenoslovje - *semantiko*, da razume kaj program pomeni in ga lahko izvaja. Sintaksa in semantika sta v programskih jezikih tesno povezani.

Sintaksa (slovnica) so pravila za zapis, kar pomeni formalizacijo in možnost avtomatskega preverjanja pravilnosti zapisa.

Semantika (pomenoslovje) daje pomen zapisu v danem jeziku, torej možnost opisovanja. Semantika določa pomen posameznih delov kode v programskem jeziku in pravilno logično zaporedje zapisa.

Če nam programska okolja omogočajo enostavno iskanje napak v zapisu, sintaksi, nam pri iskanju semantičnih ne morejo veliko pomagati, saj gre večinoma za logične napake (deljenje z 0, vsota namesto produkt, napačno zaporedje stavkov…). Iskanje logičnih napak v programu je izredno težavno, pogosto potrebujemo razhroščevalnik. Logične napake so večinoma posledica slabo pripravljenih algoritmov.

Dober programski jezik mora imeti naslednje lastnosti:

Po kriterijih uporabnika (predvsem razumljivost in obvladljivost):
Ali se je jezika enostavno naučiti in pomniti ?
Enostavnost sintakse in semantike.
Uporaba standardne notacije kjerkoli je mogoče.
Deli programov, ki izgledajo podobno se tudi obnašajo podobno.
Ali je enostavno zapisati pravilne programe ?
Ali je enostavno razumeti programe ?
Ali je enostavno spreminjati programe (neodvisnost od platforme, strukturiranost)?
Po kriterijih računalnika:
Ali ga je lahko izvajati ?
Ali je učinkovit pri prevajanju in je izvajanje hitro ?
Ali ga je moč uporabljati na različnih platformah ?

Izvajanje programov se lahko izvrši na več načinov odvisno od programskega jezika. Nekateri jeziki uporabljajo prevajanje drugi pa tolmačenje. Tolmačenje (ang. *interpreter*) je v sprotno prevajanje izvorne kode, katerega tipični predstavnik je Basic. Tolmač sproti izvaja dve aktivnosti, prevajanje in izvajanje.

Prednost tolmačenja je v tem, ker ni potrebno najprej prevajati kode in nato izvajati temveč lahko to počnemo v enem koraku, kar je dobrodošlo pri razvijanju programa. Prav tako lahko tolmači že med zapisovanjem izvorne kode preverjajo pravilnost zapisa izvorne kode. Med slabosti lahko štejemo počasnejše izvajanje (dva postopka) predvsem pri delu s podatki. Slabost je tudi, da potrebujemo za izvajanje programa na drugem računalniku tudi tolmač (enak), s katerim je bila koda zapisana.

S kakovostjo zaslonskega prikaza je prišlo do novega razvoja programske opreme v smeri objektnega programiranja. Programiranje vsakega gumba in ikone na zaslonu je izredno zamudno, zato so bili razviti programski jeziki za preprostejše programiranje in uporabo na zaslonu, ki vključujejo že obstoječo kodo za okenske funkcije (gumbi, okna, drsniki), ki jim

moramo določiti le še lastnosti (velikost, barva ipd.), hkrati pa lahko omejujemo način zapisa (oblika zapisa datuma, cene) in tega ni potrebno preverjati v sami programski kodi.

5.3 Kratek opis zgodovine razvoja programskih jezikov

Od iznajdbe računskega stroja Charls Babbage leta 1822, računalniki zahtevajo instrukcije, da izvedejo določeno nalogo. Poleg računskega stroja je razvijal tudi analitični stroj, ki ga ni uspel izdelati, ga pa danes priznavamo kot prvi stroj, ki ga lahko programiramo. Charls je v tistih letih veliko sodeloval z Ada Lovelace, hčerko Lorda Bayrona, ki mu je pri zasnovi izredno veliko pomagala, in danes velja za prvega računalniškega programerja. Po njej so poimenovali programski jezik Ada.

Programski jeziki, so bili najprej zbirka ali niz instrukcij, ki jih je bilo potrebno zapisati v pomnilnik in nato izvajati. Kasneje so začeli programski jeziki vsebovati vedno več značilnosti, kot je logično vejanje in objektno programiranje.

Računski stroj je bil mehanski in je deloval na fizični osnovi z menjavanjem zobnikov. Fizikalni princip (premiki) so bili zamenjani 1942 ko je ameriška vlada zgradila ENIAC, kjer so fizični premiki zamenjani z električnimi signali. Osnovni koncept je bil v osnovi enak, saj je bilo potrebno zamenjati ožičenje za vsak program. 1945 je John Von Neumann razvil dva pomembna koncepta, ki sta zelo neposredno vplivala na razvoj programskih jezikov. Prvi je bil »shared-program technique«, ki je zagovarjal stališče, da je strojna oprema računalnika zgrajena enostavno in tako primerna za več programov, kjer naj kompleksne instrukcije uporabljajo enostavno zasnovo, kar je omogočalo bistveno hitrejše reprogramiranje. Drug koncept, »conditional control transfer«, pa je definiral zasnovo podprogramov, ali majhnim blokom kode, ki jih lahko vključujemo v programe po potrebi. Osnovna ideja je zasnova današnjih stavkov za vejanje in podprogramov ter programskih knjižnic kode.

Leta 1949 je bil razvit programski jezik »*Short Code*«, ki je bil prvi programski jezik za elektronske stroje, ki je deloval z dvema stabilnima stanjema 0 in 1.

1957 je bil razvit prvi višje-nivojski programski jezik Fortran (*Formula Translating system*), ki ga je razvil IBM za izvajanje znanstvenih preračunov. Komponente programa so bile enostavne (IF, DO, GOTO) in so takrat pomenile velik korak naprej. Fortran je vnesel tipe spremenljivk: INTEGER, REAL in DOUBLE-PRECISION ter logične spremenljivke.

Fortran je odličen za delo s števili (numeričnimi preračuni), manj prijazen pa pri uporabi vhoda in izhoda, kar pa je bistvo poslovnih in spletnih aplikacij. Začetni programski jezik za poslovne aplikacije datira v leto 1959 s programskim jezikom COBOL, ki je bil razvit v ta namen. Uporabljal je le enostavna števila in tekst, ki jih je bilo mogoče obdelati v poljih, kar je omogočalo bistveno boljšo organiziranost podatkov. COBOL je uporabljal angleško slovnico in se ga je bilo enostavno naučiti.

Leta 1958 je John McCarthy zasnoval programski jezik LISP, ki je bil osnova za raziskave umetne inteligence in temu primerno kompleksna je sintaksa jezika. Največja razlika od ostalih programskih jezikov je, da uporablja le niz, označen z nizom znakov znotraj oklepajev in ima edinstveno možnost, da program spreminja samega sebe. 1958 je bil razvit ALGOL, katerega najpomembnejša lastnost je, da je bil osnova za nove jezike kot so Pascal, C, C++ in Java. Bil je prvi jezik, ki je imel formalno slovnico. Čeprav je vpeljal nekatere nove koncepte, kot je rekurzija, je bila njegova naslednja verzija leta 1968 težka za učenje, kar je privedlo do razvoja manjših in kompaktnejših jezikov kot je Pascal.

Pascal izvira iz leta 1968 (Niklaus Wirth) in je služil kot dobro učno orodje (editiranje) in je deloval na večini platform. Vključeval je dobre lastnosti Fortrana, Cobola in Algola, in je dobro prečiščen in čitljiv. Z dobrimi matematičnimi funkcijami in prijaznim vhodom in izhodom je postal izredno uspešen programski jezik. Pascal je izboljšal uporabo kazalcev (*ang. pointer*), ki

so izredno uporabni v vsakem jeziku, prav tako pa je vnesel stavek CASE. Pascal je pomemben tudi pri razvoju dinamičnih spremenljivk, ki jih lahko definiramo med izvajanjem programa (NEW in DISPOSE). Ni pa vključeval dinamičnih polj ali skupin spremenljivk, ki so bile vključene pozneje v jezik pod imenom Modula-2, vendar je bil razvoj prepozen saj je takrat na popularnosti že pridobival programski jezik C.

C je bil razvit leta 1972 (Dennis Ritche) in je v osnovi zelo podoben Pascalu z vsemi prednostmi tega jezika, bolj izrazita je podpora kazalcem. Jezik je bi razvit z namenom, da je močan in hiter na račun čitljivosti. Ker pa je uredil napake Pascala je programerje hitro pridobil na svojo stran. C je bil razvit hkrati z novim operacijskim sistemom Unix in se izredno dopolnjujeta. Unix je omogočil napredne funkcije kot so dinamične spremenljivke, večopravilnost in močan vhod in izhod na nizkem nivoju. Zaradi teh lastnosti je C uporabljen za razvoj operacijskih sistemov Unix, Windows, MacOS in Linux.

V poznih 70tih in začetku 80tih je bila razvita nova metoda programiranja, znana kot »objektno programiranje«. Leta 1983 je Bjarne Stroustroup nadgradil jezik C v jezik C++. Namen je bil vpeljava objektnega programiranja in obdržati hitrosti jezika C in hkrati dodati več neodvisnosti od platforme. C++ je najpogosteje uporabljan jezik za vse vrste računalniških simulacij, saj omogoča hkratno sledenje veliki količini objektov.

Leta 1994 je bila v Sun razvita Java, ki je bila najprej namenjena interaktivni TV, nato pa je bil razvoj preusmerjen za programiranje na Internetu. Najprej se je pojavila v brskalniku Netscape in hitro pridobivala na popularnosti. Čeprav je bila glede na teoretična izhodišča izredno dober jezik, je bilo izvajanje počasno zaradi problemov z optimiranjem kode. Znana je borba za Javo med Sun in Microsoft, in danes Javo razvija Sun. Java se je izkazala kot pravi objektno orientiran jezik z veliko mero prenosnosti med računalniškimi platformami.

VisualBasic je velikokrat razumljen kot prvi programski jezik, ker je njegova osnova v BASIC iz leta 1964. Basic je jezik z zelo omejenimi sposobnostmi, katerega izvajanje je zaporedno s tolmačenjem. Microsoft je Basic razvil v VisualBasic. Osnova VB je forma ali okno v katero lahko enostavno dodajamo komponente, kot so meniji, slike in podobno. Definiramo jim lastnosti kot so barva in dogodke kot je klik z miško in s tem zgradimo grafični vmesnik med uporabnikom in drugimi aplikacijami (predvsem Microsoftovimi Excel, Access). VB je danes pogosto uporabljen za hitro in enostavno kreiranje grafičnih vmesnikov v kateremkoli programskem jeziku z razmeroma malo kode.

Programski jezik Matlab se je začel razvijati v poznih sedemdesetih letih. Po univerzah je hitro pridobil veliko privržencev saj je njegova sintaksa izredno enostavna. Po letu 1984 ga je prevzela programerska hiša MathWorks in nadaljevala razvoj. Po letu 2000 je bil prekodiran z uporabo programskega jezika C, njegova osnova so postale matrične operacije. Najprej so ga prevzeli raziskovalci s področja avtomatike in vodenja elektronskih naprav. V zadnjem času se ogromno uporablja tudi za numerične analize in procesiranje slik. Ima strukturirane podatkovne tipe, ki ne potrebujejo deklaracij. Vsebuje veliko število funkcij in knjižnic za kreiranje najrazličnejših aplikacij in integrirano okolje za risanje grafov. Vključuje objekto programiranje in vključevanje funkcijskih programov zapisanih v Fortranu in C ter knjižnic zapisanih v Java in ActiveX. Matlab je licenčni program katerega konkurenta sta brezplačni okolji z zelo podobnimi lastnostmi: SCILab in Octave.

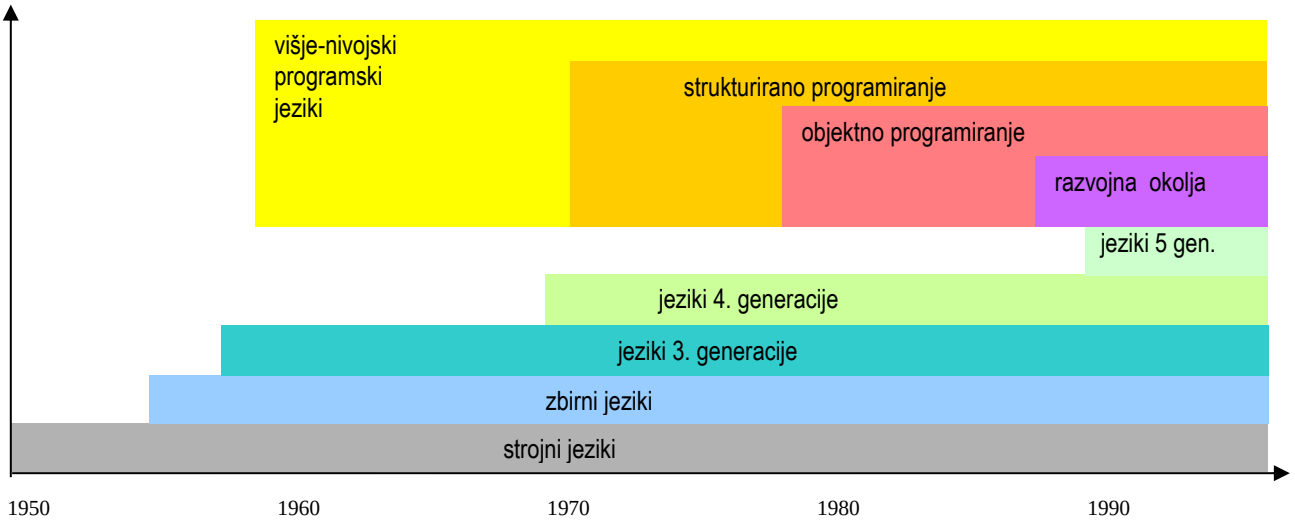
Na Internetu moramo omeniti še PERL, ker je najpogosteje uporabljen za pridobivanje in posredovanje podatkov preko spleta. Ima zelo dobre funkcije za delo s tekstom. Razvit je bil leta 1987 (Larry Wall), ko Unix še ni podpiral zajemanja in procesiranja podatkov preko spleta.

Programski jeziki se razvijajo že leta in se bodo tudi v prihodnje. Začelo se je s prvimi koraki, ko je bilo potrebno ožičiti računalnik za izvajanje določene naloge. V naslednjih korakih so nastali programski jeziki in omogočali nove in boljše lastnosti. Lastnost prvotnih jezikov je bila, da so

bili razviti za en le namen, medtem ko se današnji jeziki razlikujejo po načinu programiranja, da bi zadostili vsem namenom. Najverjetneje bodo programski jeziki prihodnosti bolj podobni naravnim jezikom na kvantnih in bioloških računalnikih.

Višje-nivojski programski jeziki premostijo razliko med uporabnikom in računalnikom s sistemom simbolov. Programski jeziki so standardno razdeljeni v zgodovinske generacije:

1. generacija: strojni (Eniac) in zbirni jezik (assembler, MacroAssembler).
2. generacija: nestrukturirani višje-nivojski jeziki, ki obvladujejo simbole, ki jih lahko prevedemo v različne strojne jezike in kjer se en programski stavek prevede v več inštrukcij strojnega jezika (Fortran, COBOL); prvi podprogrami in strukturirani višje-nivojski jeziki (Pascal, C, C++, Ada)
3. generacija: integrirani jeziki za delo s podatki (DBMS), povpraševalni jeziki in generatorji poročil (SQL), jeziki za razvoj računalniških vmesnikov.
4. generacija: visoko-nivojski jeziki, predvsem za programiranje logike, deklarativni jeziki, ekspertni sistemi, sistemi znanj, mehanizmi sklepanj, procesiranje naravnih jezikov (Prolog), matrični jeziki: Mathematika, Matlab.
5. generacija: (*angl. domain-specific languages*), opredeljuje programske jezike, ki služijo reševanju specifičnih problemov ozkega področja.



Slika 15: Časovna preglednica razvoja generacij programskih jezikov.

Splošno programske jezike delimo na več kategorij.

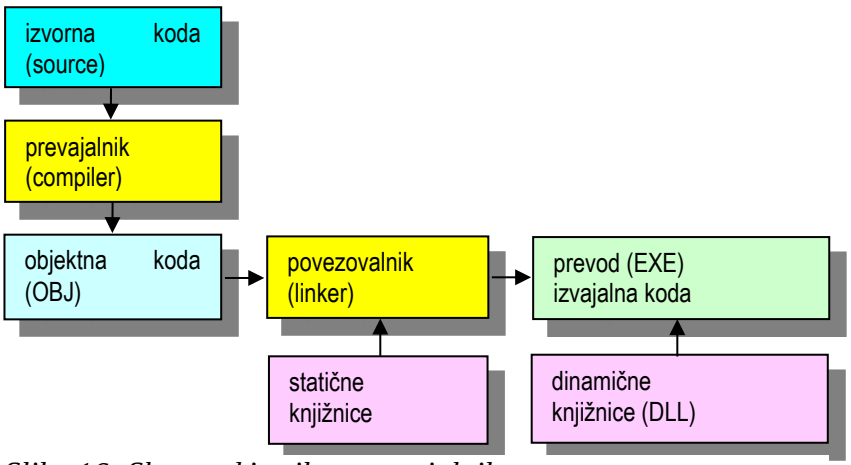
- Po namenu:
  - Jeziki za poslovne aplikacije (Cobol)
  - Jeziki za numerične aplikacije (Fortran, Algol)
  - Jeziki orientirani na sezname (Lisp)
  - Splošni sistemski jeziki (C, C++, Java, Pascal, Modula-2)

Po paradigmi programiranja:

- Jeziki za strukturirano programiranje.
- Jeziki za objektno orientirano programiranje.
- Jeziki za spletno programiranje, skriptni jeziki.

Po načinu izvajanja:

- Prevajalniki (*angl. compiler*)**:** Prevajalnik prevede celotno kodo v strojne ukaze, to kodo je potem mogoče izvajati.



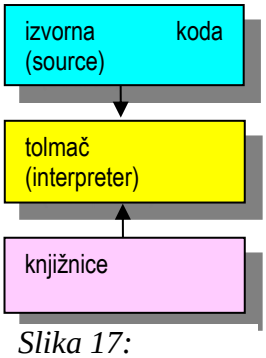
Slika 16: Shematski prikaz prevajalnika.

Lastnosti:

- čakanje na prevod po vsaki spremembi programa,
- hitro izvajanje,
- za izvajanje ne potrebujemo izvorne kode,
- zahtevnejše odkrivanje napak.

Predstavnik: Fortran, C

- Tolmači (*angl. interpreter*). Sprotno prevajanje in izvajanje posameznih ukazov programa.



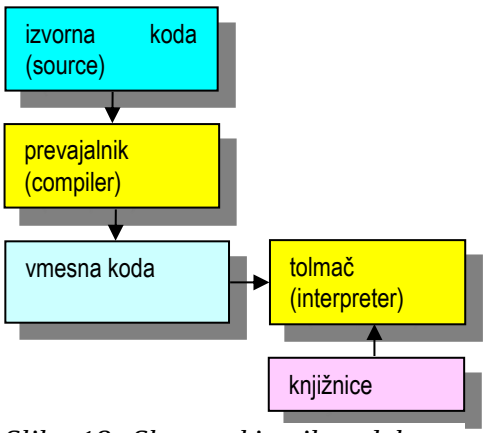
Slika 17: Shematski prikaz tolmača.

Lastnosti:

- možnost takojšnjega izvajanja po vsaki spremembi programa,
- počasno izvajanje,
- za izvajanje potrebujemo izvorno kodo in tolmač,
- lažje odkrivanje napak v programu,
- možnost "samospreminjanja" programa.

Predstavnik: Basic, Matlab

- Kombinacija: Delno prevajanje v vmesni jezik; izvorno kodo prevedemo v nek standardizirani jezik, ki ni strojni.



Slika 18: Shematski prikaz delnega prevajanja.

Lastnosti:

- ga je treba ob izvajanju interpretirati,
- lažje, standardizirano prevajanje,
- srednje hitro izvajanje,
- vmesna koda je neodvisna od računalnika,
- prenosljivost na nivoju vmesne kode.

Predstavnik: Java

Razvoj programskih jezikov je tudi posledica krize pri programiranju obsežnejših programov in programske opreme (OS). Vzroki za premajhno produktivnost programiranja:

- cena strojne opreme se zmanjša za nekaj tisoč krat.
- produktivnost pri razvoju programske opreme se poveča za pet krat.
- kompleksnost aplikacij je veliko večja (včasih s klasičnimi orodji neobvladljiva).
- zbranih podatkov, ki so potrebni računalniške obdelave, je vse več.
- premalo dobrih programerjev, šolanje je počasno in drago.

Povečanje produktivnosti programiranja je mogoče doseči bodisi z enako kakovostjo produktov v krajšem času, učinkovitost posameznika mora biti večja ali hkratno povečanje tako kakovosti kot učinkovitosti. Za večjo produktivnost morajo programske jeziki imeti naslednje lastnosti:

- poudarek na cilju, rezultatsko orientirani,
- enostavnost za učenje in uporabo,
- manjši čas načrtovanja in čas kodiranja,
- preverjanje funkcionalnosti, testiranje pred končanim razvojem,
- enostavnost širitve in prilagajanja,
- enostavnejše operacije lahko razvijejo uporabniki sami,
- zmanjšana kompleksnost razvoja.

Pri razvoju programskih jezikov ne smemo izpustiti skriptnih jezikov. Skriptni jeziki omogočajo zaporedno zapisovanje ukazov programskega okolja z dodanimi funkcijami za branje parametrov in krmiljenje toka izvajanja programov. Uporabo skriptnih jezikov je najlaže ponazoriti na jezikih Java in JavaScript. Za Javo že vemo, da je sistemski programski jezik, ki je neodvisen od platforme. Java program prevedemo v binarni psevdojezik Java VM (*Virtual Machine*), ki omogoča hiter prenos preko mreže in hitrejše izvajanje. Za samo izvajanje je v vsakem operacijskem sistemu poseben modul, ki tolmači psvedojezik, ki pa je počasnejši.

Tipični in najbolj znan skriptni jezik je JavaScript, znana pa sta tudi Microsoftova VBScript (temelji na VisualBasicu) in JScript. Skriptni jeziki se uporabljajo v programiranju makrov, večinoma pa so vključki v HTML straneh. Ti skriptni jeziki so namenjeni predvsem delu na spletnih straneh, in zato ne potrebujejo posebnega prevajalnika, saj za prevajanje poskrbi sam brskalnik. Skriptni jeziki se med seboj ločijo večinoma v sintaksi in možnostih vključevanja funkcij operacijskega sistema in spletnega brskalnika. Skriptni jeziki pri spletnih aplikacijah se izvajajo na računalniku uporabnika in niso podvrženi hitrosti delovanja omrežja. Ne smemo pozabiti, da je Java sistemski programski jezik JavaScript pa skriptni jezik, ki je namenjen kot dodatek HTML.

6 PROGRAMIRANJE

Programiranje je reševanje problemov.

Dobro in logično programiranje je proces razvoja programa, ki se prične s planiranjem in organiziranjem dela. Vsak problem lahko rešimo z izvajanjem niza aktivnosti v določenem vrstnem redu, algoritmu. Izkušnje dokazujejo, da je najtežji del programiranja problema na računalniku prav razvijanje algoritma za reševanje. V tem poglavju je večina programskih rešitev zapisana z algoritmom predstavljenim z diagramom poteka in kodo v štirih programskih jezikih. Prikazovanje kode v več programskih jezikih ima svoj namen: ugotovimo lahko enakost logike in načina reševanja problemov in majhno razliko v sintaksi programskih jezikov.

Programiranje je za ene znanost, za druge umetnost, za večino programerjev pa je obrt, s katero se preživljajo. Obrti se ne da naučiti brez prakse in tako je tudi s programiranjem. Če želimo s pomočjo računalnika rešiti nek problem, moramo sestaviti postopek, ki ga bo računalnik razumel. Pisanje programa ali programiranje je torej proces ali postopek reševanja problema z računalnikom, ki vsebuje več faz:

- definicija problema in primernost za reševanje z računalnikom (kateri so podatki in kakšen rezultat želimo),
- definicija algoritma (načrtovanje postopka rešitve; razgraditev na zaporedje navodil postopka),
- kodiranje v programskem jeziku, ki jih zmore računalnik izvesti,
- izvajanje programa in
- testiranje programa (ali so rezultati pričakovani).

Že iz same definicije problema sledi, da ne moremo zapisati programa, če problema (za vsaj enostaven primer) ne zmoremo rešiti brez računalnika (lahko pa uporabimo kalkulator). Za enostavne probleme večinoma ne pišemo algoritmov temveč jih kodiramo direktno. Brez algoritma je pri kodiranju več nejasnosti in posledično več testiranja in spreminjanja programske kode.

Programiranje v najožjem smislu torej pomeni zapis kode v programskem okolju ali pa v kateremkoli urejevalniku besedil, in nato prevajanje in izvajanje in testiranje kode.

Ker bomo v tem poglavju obravnavali štiri pogoste programske jezike jih najprej na kratko predstavimo: Fortran, Java, VisualBasic in Matlab.

Fortran

Je eden najstarejših višje-nivojskih programskih jezikov, ki je bil razvit pri IBM pod vodstvom Johna Backusa leta 1957. Ime jezika je akronim za **FOR**mula **TRAN**slation, ker je bil razvit za enostavno pretvorbo matematičnih formul v programski jezik.

Fortran je sinonim za postopkovni programski jezik, ki se uporablja za programiranje numerično zahtevnih znanstvenih aplikacij. Bil je prvi višje-nivojski jezik, ki je uporabljal sistem s prevajanjem. Nameni razvoja jezika so bili: enostavnost za učenje, primernost za različne aplikacije, neodvisnost od računalniške platforme, izražanje kompleksnih matematičnih izrazov podobno matematični algebrajski notaciji. Glede na stanje programiranja v času nastanka Fortrana je ta jezik pospešil kodiranje za 500%, sama učinkovitost izvajanja glede na zbirni jezik pa je padla le za 20%, kar je omogočalo programerjem večjo skoncentriranost na reševanje problemov kot na kodiranje.

Razvoj programskih jezikov je leta pripeljal do standardizirane verzije, Fortran77. Zaradi razvoja objektno orientiranih programskih jezikov je bil Fortran zopet posodobljen leta 90 in 95, tako da je zadnja standardna verzija Fortran95. Hkrati pa je bil razvit tudi prvi standardni jezik za večprocesorske in paralelno sestavljene računalnike, ki se imenuje HF – Hyper Fortran, in leta

2003 standardizirana verzija Fortrana2003, ki omogoča objektno orientirano programiranje in razmeroma enostavno vključevanje programskega jezika C neposredno v kodo Fortran.

Lastnosti Fortran

- Enostavno učljiv: enostavno se ga je naučiti in razumeti.
- Neodvisen od platforme: zaradi standardiziranosti enostaven prenos med računalniki.
- Naraven način opisa matematičnih funkcij, saj je mogoče tudi kompleksne matematične funkcije izraziti podobno kot v matematičnih algebrskimi izrazih.
- Problemsko orientiran.
- Dobro izkorišča strojno opremo.
- Izredno učinkovit (hiter) pri izvajanju kode (še vedno le 20% počasnejši kot zbirni jezik).
- Dobra kontrola nad pomnilnikom.

Java

Java je sistemski programski jezik, katerega začetki segajo v leto 1990 najprej kot Oak pri Sun Microsystems z namenom voditi interaktivno TV. Interaktivna TV ni zaživela, izraziti vzpon pa doživel svetovni splet (Web) in leta 1994 je izšel HotJava z možnostjo programčkov (*ang. applet*) na spletu. 1996 je izšla prva verzija JDK (Java Development Kit), leta 1997 verzija 1.1 in leta 1999 verzija 1.2, ki so jo poimenovali Java2 z razredi in izboljšanimi vmesniki za računalniško grafiko (*awt, swing*). Sun je dosegel, da je bila standardizirana pod njegovim imenom (tudi MicroSoft je imel Javo). Java je kreirana tako, da se izvaja na Javinem navideznem stroju (*ang. Java Virtual Machine*), ki izvaja kodo zapisano v Java. Kodo je potrebno najprej prevesti v binarno kodo, ki jo nato Javin navidezni stroj tolmači na računalniku. Javini programčki (*ang. applet)* se izvajajo znotraj spletnega brskalnika. Torej mora imeti brskalnik že vgrajen Javin virtualni stroj, ali pa je tega potrebno dodati kot vtič (*ang. plug-in*).

Lastnosti Java:

- Je enostavnejša kot C ali C++ (nima kazalcev, predprocesorja) vendar je po strukturi zelo podobna in programerjem adaptacija ni povzročala težav.
- Je objektno orientirana.
- Je kombinacija prevajalnika in tolmača.
- Je robustna in zahteva natančni zapis (vse spremenljivke je potrebno deklarirati).
- Zelo neodvisna od platforme (ne pa popolnoma) in varna, programe je možno izvajati na različnih operacijskih sistemih.
- Sorazmerno učinkovita (povprečno dosega 50% pa včasih tudi 100% učinkovitosti kode zapisane v C++).
- Omogoča izvajanje več procesov hkrati (ang. *threats*), kar je izrazito uporabno v večpredstavnostnih aplikacijah.
- Primerna je tako za obsežne aplikacije kot za majhne programčke in vmesnike na mobilnih enotah (telefoni).
- Slabosti pri uporabi numerično orientiranih aplikacij:
- Simulacije fizičnega sveta:
    - Objekti - simulacija logičnih modelov iz narave
    - Niti – objekti se lahko izvajajo hkrati kot je to v naravi
    - Prikaz grafike – močna grafična podpora za prikaz simulacij
- Delo preko spleta – Java omogoča tudi nepoznavalcem spleta razmeroma enostavno kodiranje spletnih aplikacij za spremljanje in vodenje eksperimentov.
- Dostop do podatkov in analiza preko spleta – programčki (ang. *applets*) so učinkovito orodje za vmesnike do velikih podatkovnih baz preko spleta, lahko so tudi analize, grafi, slike in podobno.

- Programčki omogočajo prikaz fizičnih procesov in omogočajo interaktivnost, kar je dobrodošlo v procesu šolanja in treninga.
- Izredna možnost souporabe že napisane kode za vključevanje v različne aplikacije na različnih platformah.

VisualBasic

VisualBasic ima od opisanih programskih jezikov krajšo, a zanimivo zgodovino. Začetki VisualBasic segajo v leto 1987, ko je bil izdan pri Microsoft, ki v samem startu ni bil popularen in razširjen. Z verzijo 2 in 3 pa je postal programski jezik, ki se na trgu jezikov zelo hitro širil.

Pred VisualBasic se je za razvoj aplikacij v okolju Windows uporabljal C ali C++, kar je pomenilo precej kompliciran proces. Leta 1991 je Microsoft izdal VisualBasic 1, ki je omogočal enostavno in hitro kreiranje aplikacij, z osnovno zamislijo, da program reagira na dogodke, ki so sproženi s strani uporabniškega vmesnika. Leta 1992 je izšel VisualBasic 2, ki je vpeljal forme in objektne spremenljivke, leta 1993 VisualBasic 3, ki je omogočal dostop do podatkovnih baz, leta 1996 VisualBasic 4, ki je vpeljal razrede in omogočal uporabnikom dodajanje vtičev (ang. *add-inns*), leta 1997 VisualBasic 5, ki je omogočal povezave z Windows 95 in ActiveX in 1998 VisualBasic 6, ki je vpeljal ActiveX Data Objects (ADO) za delo s podatki, kreiranje podatkovnih baz za aktivne strani na strežnikih. Leta 2002 je bil predstavljen VisualBasic.NET. VisualBasic.NET je objektno orientiran programski jezik, ki omogoča uporabo strežnikov (ang. *Framework*) in Web Services kot so SOAP, ki uporabljajo XML standard. VisualBasic.NET ima popolnoma novo zasnovo in ni kompatibilen s VisualBasic 6. Jezik je namenjen kreiranju kode za spletne aplikacije (.NET applications, Windows ali Web applications, in Web Services.). VisualBasic.NET ima GUI, ki omogoča enostavno dodajanje objektov, ki jim predpišemo parametre in omogoča enostavno komunikacijo s strojno opremo. Danes lahko rečemo, da je to najprimernejši jezik za programiranje aplikacij v operacijskem jeziku Windows. Podobni programski okolji sta Visual C in Delphi.

Lastnosti VisualBasic:

- Ni samo programski jezik, temveč programsko okolje, ki uporabniku z manj izkušnjami omogoča razvoj aplikacij (posebej v povezavi z Microsoft produkti kot je Excel). Osnovni namen je »*Personal programming*«, kjerkoli in karkoli delamo lahko obstoječe aplikacije personaliziramo z dodajanjem ali spreminjanjem kode.
- Enostavnost kreiranja programov z uporabo grafike (GUI) – avtomatizacija (v večini aplikacij potrebujemo 80% časa za razvoj vmesnika). Idealen je za delo z Windows.
- Primernost za razvoj aplikacij kot pred in po procesorji pri uporabi podatkovnih baz (SQL)
- Delitev na *Form* kot standardne okenske aplikacije, ki jim predpišemo lastnosti in parametre, in *controls*, s katerimi dodajamo dogodke pri katerih se izvedejo aktivnosti.

VisualBasic je pravzaprav okolje za razvoj programov (tudi spletnih) in zapis kode v operacijskem sistemu Windows. Tudi kompleksne programe je mogoče razviti razmeroma hitro, saj imamo na razpolago veliko število funkcij za grafično predstavitev in uporabo OS. Deluje na sistemu dogodkov "*event-driven*", kar pomeni, da lahko procedure kličemo avtomatsko pri izbiri iz menijev, kliku miške, premik objektov po zaslonu in podobno. Kot že omenjeno je programsko okolje, ki omogoča uporabo knjižnice velikega števila funkcij, ki jih ni potrebno programirati temveč jim samo predpišemo parametre in vključimo v programsko kodo. Lastnost tako zapisanih programov je tudi v navzven enakem izgledu programov, v primerjavi z aplikacijami.

Matlab

Matlab uvrščamo med programske jezike četrte generacije in predstavlja izredno močno in učinkovito orodje, ki se je uveljavilo kot standard za simulacijo, analizo podatkov in numerično računanje v industrijskem, znanstvenem in izobraževalnem okolju. Predstavlja integrirano tehnično računsko okolje, ki združuje numerično računanje, napredno grafiko in vizualizacijo ter višji programski jezik. To omogoča uporabniku enostaven in hiter zapis in testiranje novih zamisli ter razvoj novih aplikacij za hitro doseganje praktičnih rezultatov.

Temelje za Matlab postavil Cleve Moler v osemdesetih letih prejšnjega stoletja, ko je zapisal v programskem jeziku Fortran prve funkcije za računanje lastnih vrednosti matrik in reševanje linearnih sistemov. Jack Little and Steve Bangert sta prepoznala potencial Matlab, prevedla funkcije v C programski jezik, dodala uporabniško definirane funkcije in izboljšala grafično okolje. Skupaj so vsi trije leta 1984 ustanovili podjetje Mathworks, ki je ponudilo prvi programski paket PC Matlab 1.0 za takrat nove IBM-ove osebne računalnike. Od takrat je podjetje razvilo in ponudilo na tržišču več verzij programa, ki jih je dopolnjevalo z dodatnimi orodji (toolboxes) in zadnja, ki se ponuja, je verzija 8.1.

Dodatna, aplikacijsko orientirana orodja, razširjajo Matlab-ovo okolje in omogočajo reševanje specifičnih problemov. Ta orodja so razvili eksperti teh področij, uporabnikom pa dajejo možnosti učenja, uporabe in primerjave naprednih tehnik ter ocenjevanja različnih pristopov brez pisanja programske kode. Matlab se najpogosteje uporablja za matematične izračune, razvoj algoritmov, modeliranje, simulacijo in analizo, za obdelavo podatkov in vizualizacijo, grafiko in razvoj aplikacij, vključujoč tudi izdelavo grafičnih uporabniških vmesnikov.

Med prednosti Matlab prištevamo interaktivno delo, jedrnato sintakso in vgrajene operatorje, ki omogočajo hitro in enostavno programiranje, prav tako dodana grafična podpora omogoča enostavno, hitro in kvalitetno vizualizacijo. Ena pomembnejših odlik Matlab programskega orodja je njegova odprta struktura, ki omogoča enostavno prenašanje podatkov med Matlab-om in ostalimi aplikacijami ter neodvisnost programske kode in podatkovnega formata od operacijskega sistema.

Matlab programski jezik prištevamo med tolmače, posledično se kot slabost kaže počasnejše izvajanje programov. Kot slabost pa lahko označimo tudi dejstvo, da ni prosto dostopen in da je njegova nakupna cena visoka.

Dandanes velja Matlab za najbolj razširjen programski jezik oziroma orodje za tehnično računanje in ga po celem svetu uporablja že več kot milijon uporabnikov.

Lastnosti Matlab:

- Tipičen proceduralni jezik.
- Ima vse glavne osnovne programske konstrukte kot C, Java, …
- Enostaven za uporabo. Hitro učenje.
- Priročno programsko okolje.
- Osnova je matrika (spominja na programe za delo s preglednicami)
- Tolmač (interpreter), lažji za začetnike, lažje iskanje in popravljanje napak.
- Počasen pri izvajanju.
- Interaktivno izvrševanje ukazov.
- Hitro pisanje prototipnih programov.
- Enostavna vizualizacija.
- Ni potrebno definirati podatkovnih tipov. Ni splošen programski jezik

6.1 Osnovni pojmi zapisa kode v programskih jezikih

To poglavje opisuje osnovne pojme kodiranja, zapisane, kolikor je to mogoče, neodvisno od programskega jezika. Vsi programski jeziki, kot posledica zgodovinskega razvoja in glede na dejstvo, da delujejo na enaki ali podobni platformi, povzemajo določene lastnosti, ki jih računalnik kot naprava ima. Programske jezike sestavljajo opisi:

- podatkov:
  - konstante in
  - spremenljivke
- ukazov:
  - prireditve in izrazi,
  - zajemanje podatkov in izpis,
  - kontrola in vodenje programa (vejitev in zanke)
- strukture:
  - funkcije, metode in podprogrami,
  - objekti in podatkovne strukture.

Kot osnovni pojmi so opisani deli programskih jezikov, ki služijo zapisu v pomnilniku in vodenju izvajanja programa in se pojavljajo v programskih jezikih z različno sintakso imajo pa enako funkcijsko namembnost.

Programe zapisujemo v obliki stavkov. Stavke sestavljajo ukazne besede, operatorji, ločila in podatki. Ukazne besede in oblika zapisa stavkov so predpisane v sintaksi programskih jezikov. Kot tipični primer lahko ponazorimo enostavni stavek za vejenje. Sestavljen je iz ukazne besede **IF** zapisa pogoja (spremenljivka **n**, operator > in konstanta **0**) in izraza (spremenljivka **x**, operator – in ukaz za prireditev vrednosti =), ki se ob tem pogoju izvrši. Sam zapis se po sintaksi v programskih jezikih razlikuje, kar je nazorno prikazano na primeru:

```
Fortran          IF (n>0) x=-x
VisualBasic      if n>0 then x=-x
Java             if (n>0) {x=-x}
Matlab           if n>0 x=-x, end
```

Zapis kode je odvisen od programskega jezika, ki ga uporabljamo. Fortran ne razlikuje med velikimi in malimi črkami zapisa (`imeSpremenljivke=IMESPREMENLJIVKE=imespremenljivke`), medtem, ko je v VisualBasic v veljavi tako imenovani kamelji zapis (`ImeSpremenljivke`), ki se prične vedno z veliko začetnico, v Java velja dogovor, da se imena razredov piše z veliko začetnico ostalo pa z malimi črkami, telo razreda je zapisano znotraj {}. Matlab loči male in velike črke in ne potrebuje deklaracije spremenljivk. Vse spremenljivke so zapisane v matrični obliki, v obliki polj. Navadna spremenljivka je polje z eno vrstico in enim stolpcem.

Komentarji v programski kodi služijo za komentiranje kode. Čeprav je program sam po sebi najboljši komentar, če je zapisan čitljivo in jasno, je potrebno kakšne prijeme (posebnosti) v programski kodi posebej dokumentirati. Ti komentarji služijo za razumevanje programske rešitve, predvsem pri kasnejšem spreminjanju programov. Pri zapisu programske kode moramo vedno misliti na to, da jo bo potrebno prilagajati in popravljati. Večina programerjev na začetku programa v komentar zapiše namen programske kode, ime avtorja, datum zadnje spremembe programa, spisek uporabljenih spremenljivk, posebne lastnosti spremenljivk in podobno. Komentarji so v različnih programskih jezikih označeni različno, ločimo pa predvsem dva tipa komentarjev: tiste, ki imajo oznako za začetek in konec komentarja (komentar je zapisan v bloku) in tiste, ki imajo za oznako le začetek komentarja, ki se avtomatično konča s koncem vrstice. Programi pri izvajanju komentarje ignorirajo.

Indikator nadaljevanja v programski kodi uporabljamo kadar je celoten ukaz predolg za eno vrstico. Kot primer lahko služi kompleksna formula ali veliko teksta za izpis. Nadaljevanje

vrstice uporabljamo torej zato, ker je zapis v vrstici zelo dolg (omejitev dolžine ene vrstice se giblje večinoma okrog 132 znakov) ali pa, kar se zgodi večkrat, zaradi čitljivosti zapisa.

V nekaterih programskih jezikih moramo označiti zaključek stavka. V večini programskih jezikov je to podpičje. V Fortran je potrebno konec stavka označiti le, če je v eni vrstici več stavkov. V programskih jezikih, ki z podpičjem označujejo konec stavka, lahko za podpičjem prosto pišemo komentar. Podpičje na koncu stavka v Matlabu pomeni, da se vrednost spremenljivke ne izpisuje in v vrstico lahko zapišemo več stavkov.

6.2 Zapis podatkov

6.2.1 Konstante

Konstante so določene vrednosti, večinoma števila (cela, realna ali kompleksna), logični vrednosti (pravilno *true* in nepravilno *false*) in zaporedja znakov ali nizi (*angl. string*). Realna števila zapisujemo z **decimalno piko** in ne vejico. Nizi so v programski kodi zapisani med znakoma, večinoma sta to " ali ', kjer je potrebno poudariti, da je začetek in konec niza označen z enakim znakom (v VisualBasic se uporablja le ", ker je znak ' uporabljen za komentarje). Če je v nizu zapisano število ne moremo z njim računati, saj je v tem primeru le niz znakov oziroma števke. Tako niz znakov "123" pomeni le niz števk 1, 2 in 3. Prav tako je lahko niz "1+2=3", ki ni z ničemer povezan z operacijami nad števili.

Torej ločimo cela, realna in kompleksna števila, niz znakov in logični konstanti.

Tip konstant		**primeri**
celo število	`integer,int`	`1 -999 0 32767`
realno število	`real, float`	`3.14159 0.123E+02`
kompleksno število	`complex`	`(2.0,3.0)` → 2+3i
niz	`string, character`	`'Besedilo' "+69"`
logična	`logical`	`.TRUE. .FALSE.`

6.2.2 Spremenljivke

Med izvajanjem programov večinoma podatke preberemo in nad njimi izvajamo operacije, vrednosti, ki jih pridobimo med izvajanjem programa se lahko spreminjajo in torej niso konstantne. Torej potrebujemo možnost zapisa vrednosti, ki jo lahko v programu prikličemo kadarkoli jo potrebujemo in ji lahko spremenimo vrednost. Zato imajo spremenljivke imajo svoje ime, na katerega se v programu sklicujemo, svoj tip in vrednost. Ime spremenljivke se mora začeti s črko anglosaške abecede, dolžina imena pa je večinoma omejena na 32 znakov.

Za razumevanje delovanja programov moramo zagotovo poznati zapis vrednosti v pomnilniku računalnika. Najbolj enostavna ponazoritev (pa še vedno dovolj verodostojna) bi bila, da si polje za zapis v pomnilniku predstavljamo kot zaključeni prostor, ki je razdeljen na tri dele (ne ukvarjamo z različnimi dolžinami in oblikami zapisa). Prvi del vsebuje kazalec za to polje na katerega se računalnik sklicuje, ko potrebuje podatke iz tega polja. Drugi del polja zaseda ime spremenljivke, na katerega se sklicujemo v programu, tretji del polja pa vrednost spremenljivke. Kar je tukaj najbolj očitno je to, da je pod enim imenom zapisana ena vrednost. Pri programiranju se večinoma sklicujemo na ime spremenljivke, da pridobimo vrednost zapisa. Pri zapisu je očitno pomembno kakšno vrednost želimo zapisati v polje: numerično (celo, realno), logično ali niz.

Imena spremenljivk naj bodo *mnemonična*, kar pomeni, da naj čim več povedo o spremenljivki sami in njenem namenu. Imena spremenljivk so lahko sestavljena iz anglosaških črk, torej ne morejo vključevati slovenskih črk: Š,Č in Ž. Imena spremenljivk ne smejo biti enaka rezerviranim besedam v programski kodi - semantika (npr: IF, TRUE). Znotraj ene programske enote na moremo imeti več spremenljivk z enakim imenom. V imenu spremenljivke tudi ne smejo biti posebni znaki in ločila, ki jih uporabljajo programski jeziki (> ; , .]) !). Tudi kadar je ime spremenljivke dolgo je nečitljivo. Omeniti še velja, da grških črk ali matematičnih simbolov, kot na primer: α,π,ω,δ,∑, ne moremo vtipkati neposredno preko tipkovnice in jih tudi ne moremo uporabljati v imenih spremenljivk. Tipi spremenljivk so enaki kot pri konstantah in dejansko pomenijo tudi način zapisa v pomnilniku. Spremenljivko lahko med izvajanjem programa uporabimo in njeno vrednost spreminjamo poljubno mnogo krat.

Ker so lahko vrednosti spremenljivk različne, moramo v programu definirati kakšen tip vrednosti bo v spremenljivki zapisan, saj je od tega odvisna oblika zapisa in dolžina polja zapisa. Zato moramo vse spremenljivke v programski kodi deklarirati. Deklaracija pomeni, da računalniku definiramo (predpišemo) kakšna vrednost se bo v spremenljivki nahajala in zanjo rezervira primeren prostor v pomnilniku. V Fortran, ki je strogo tipiziran programski jezik, moramo vse spremenljivke deklarirati na **začetku** programske enote (za stavkom, ki definira začetek programske enote), v JAVA, C/C++ pa **kadarkoli** pred uporabo spremenljivke. V nekaterih jezikih spremenljivk ni potrebno deklarirati (Matlab), vendar nam nedeklarirane spremenljivke pri izvajanju z operatorji lahko povzročijo nepredvidljive rezultate.

V spodnjem primeru imamo zapis treh spremenljivk, cele, realne in niza.

Povsem jasno je, da je lahko v eni spremenljivki shranjena le ena vrednost. Če vrednost spremenljivke med izvajanjem programa spremenimo, je nova vrednost prepisana preko stare in je stara vrednost zbrisana in nedosegljiva.

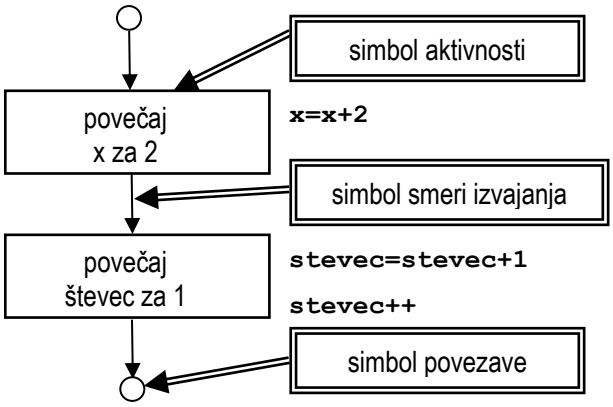
6.3 Prireditve in izrazi

Če želimo spremenljivki spremeniti vrednost to opravimo s prireditvijo vrednosti ali izrazom. Prireditev je najenostavnejša oblika izraza, ki ima eno izjemo. Prireditev začetne vrednosti spremenljivke lahko uporabimo tudi v deklaraciji spremenljivke, izraza pa ne. Način kako izvedemo prireditev, pa tudi izraz, je enaka v vseh programskih jezikih in ima obliko:

```
ImeSpremenljivke = vrednost/izraz
```

Slika 12 prikazuje diagrama poteka za prireditveni stavek.

Na levi strani zapišemo ime spremenljivke, katere vrednost je vrednost izraza na desni strani. Programski jeziki izvedejo izraz na desni in pridobljeno vrednost zapišejo kot vrednost spremenljivke. Logično je, da mora tip vrednosti izraza odgovarjati tipu spremenljivke.



Slika 12: Diagram poteka prireditvenega stavka.

Na primer: `X=2` v spremenljivko `X` zapiše vrednost `2`.

Ker je lahko na levi strani prireditve le ime spremenljivke ne moremo zapisati: `Y+1=X-2` ali `1=Y+1`.

Prireditev vrednosti spremenljivkam je zelo podobna v vseh programskih jezikih:

Tabela 3: Prireditev vrednosti spremenljivki.

Tip spremenljivke	Fortran	Java	VisualBasic	Matlab
Celoštevilčna	`x = 1`	`x = 1;`	`x = 1`	`x = 1`
Realna	`pix = 3.14`	`pix = 3.14;`	`pix = 3.14`	`pix = 3.14`
Niz	`ime="Janez"`	`ime = "Janez";`	`ime = "Janez"`	`ime = 'Janez'`
Logična	`yn = .true.`	`yn = true;`	`yn = True`	`yn = true`

Prireditev vrednosti spremenljivk je odvisna od tipa spremenljivke v katero shranimo vrednost. Če je spremenljivka `y` deklarirana kot realna lahko zapišemo `y=1` in se kot vrednost spremenljivke zapiše realna vrednost `1.000000`. Če pa napravimo obratno, v spremenljivko `z`, ki je deklarirana kot celoštevilčna poskusimo zapisati realno vrednost, na primer: `z=1.99`, bo zapisana vrednost spremenljivke `z` enaka `1`, saj bo decimalni del odrezan (ne zaokrožen kot bi utegnili misliti). Iz tega sledi, da je pomembno kakšne vrednosti želimo imeti v spremenljivki in jo na takšen način deklariramo. Programski jeziki dopuščajo spremembe tipov spremenljivk na različne načine, v nekaterih je to enostavno (Perl) v drugih pa je potrebna uporaba posebnih funkcij. Posebnost je Matlab, ki nima celoštevilčnega zapisa spremenljivk, cele vrednosti spremenljivk zapisuje kot realne in jih lahko izpišemo kot cele (dodatne funkcije za zaokroževanje).

Izraz je podoben prireditvi in ni **enačba**. Na levi strani enačaja imamo zapisano ime spremenljivke, na desni strani pa izraz, ki ga sestavljajo konstante, spremenljivke in različni operatorji.

Pri računalniškem programiranju je zelo pogost izraz povečanja spremenljivke, na primer: `x=x+1`, ki nima logične matematične osnove, zelo nazorno pa ponazarja način dela:

`x=x+1` pomeni, da računalnik vzame vrednost, ki je shranjena v spremenljivki `x` jo poveča za ena in ponovno zapiše v spremenljivko `x`. V diagramih poteka se je uveljavila drugačna notacija zapisa `x←x+1`.

Če je bila vrednost `x` pred operacijo `2` je po tej operaciji njena vrednost `3`. Kot rečeno, je ta operacija zelo pogosta, zato je v nekaterih programskih jezikih predpisan tudi krajši zapis s pomočjo prireditvenih operatorjev.

Tabela 4: Aritmetični operatorji.

Aritmetični operatorji	Fortran	Java	VisualBasic	Matlab
potenciranje	**			^
množenje	*	*	*	*
deljenje	/	/	/	/
seštevanje	+	+	+	+
odštevanje	-	-	-	-

Znak minus lahko predstavlja razen odštevanja tudi negacijo, ki ga uporabimo za spreminjanje predznaka spremenljivke. `A=-A`

Ne smemo pozabiti, da množenje označujemo z zvezdico * in ne kot v matematiki, kjer ga lahko izpustimo (presledek) ali pogosto označimo z znakoma • ali ×. Prav tako med osnovne funkcije spadajo oklepaji s katerimi definiramo vrstni red operacij. Prednost izvajanja operacij je enaka kot v matematiki in enako lahko izvajanje spreminjamo z uporabo oklepajev.

Vrednost spremenljivke **a** se po prireditvenem stavku: `a = x+y-2/2+z;`

zelo razlikuje od stavka: `a = x+(y-2)/(2+z);`

Iz zgornje tabele osnovnih aritmetičnih operatorjev je razvidno, da Java in VisualBasic nimata operatorja za potenciranje. Problem rešimo z množenjem ali kreiranjem razredov za to funkcijo. Novejši programski jeziki, ki so vzeli za osnovo programski jezik C, imajo skrajšane prireditvene operatorje.

Skrajšani prireditveni operatorji

Večanje: operator **++** poveča vrednost spremenljivke za 1; in manjšanje: operator **--** zmanjša vrednost spremenljivke za 1.

Operatorja lahko uporabimo pred ali po spremenljivki, če je pred spremenljivko najprej poveča vrednost spremenljivke in jo nato uporabi, če je operator za spremenljivko, pa jo najprej uporabi in nato poveča. Ta možnost je izredno dobrodošla v zankah.

Primer: stavki se izvajajo zaporedno

```
i=5                                    i=5
A=i++; // vrednost A=5, i=6            A=i--; // vrednost A=5, i=4
B=++i; // vrednost B=7, i=7            B=--i; // vrednost B=3, i=3
```

Osnovne matematične operacije so prisotne v vseh jezikih, omeniti pa velja, da določene operatorje uporabljamo v različnih jezikih tudi za druge operacije.

6.4 Zajemanje in prikaz podatkov

Podatke lahko med izvajanjem programa vnašamo preko tipkovnice ali miške ali jih preberemo iz datoteke. Iz datotek beremo podatke, če jih dobimo iz drugih aplikacij, ali če dostopamo do podatkovnih baz. Tudi rezultate izvajanja programa večinoma izpisujemo na zaslon ali v datoteke. Na zaslon izpisujemo tudi delne rezultate ali obvestila, opozorila med izvajanjem programov. V datoteke rezultate izpisujemo, če jih še potrebujemo za nadaljnjo obdelavo v drugih aplikacijah. Programski jeziki za vnos podatkov in izpis rezultatov uporabljajo raznolike rešitve. V poglavju je bila omenjena primernost programskih jezikov za različne vrste aplikacij, kjer je bilo predstavljeno, da na primer Fortran uporablja zelo enostavne metode za izpis na zaslon, medtem ko sta Java in VisualBasic bistveno bolj prijazna na tam področju, saj uporabljata že definirane objekte (**`MsgBox, DialogBox`**).

S stavki za branje podatkov dosežemo, da program vnesene podatke shrani v pomnilnik kot spremenljivke. Oblike teh stavkov so v različnih programskih jezikih izredno različne, skupna lastnost pa je, da pričakujejo vnos preko tipkovnice zaključen s tipko *Enter*. Seveda mora biti vnesena vrednost skladna s tipom prebrane spremenljivke, razen pri Matlab, ki ne potrebuje deklaracije spremenljivk. V celoštevilčno spremenljivko ne moremo prebrati niza ali realnega števila. Če želimo podatke brati iz datoteke ali jih zapisati v datoteko, jo je potrebno pod nekim imenom (na primer **`ime.dat`**) najprej odpreti. Hkrati velja omeniti, da na primer Java ne bere neposredno različnih tipov spremenljivk, temveč prebere zapis kot niz znakov, ki ga je potrebno nato razčleniti (*angl. parse*) na spremenljivke različnih tipov.

Pisanje in branje na zaslon in s tipkovnice:

Fortran:

```
INTEGER::a
PRINT*,' Vnesi a:'
READ*,a
PRINT*,' vrednost= ',a
```

Java:

```
String s = null; int a;
System.out.println(" Vnesi a:");
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
s = br.readLine(); a = Integer.parseInt(s);
System.out.println("vrednost= " + a);
```

VisualBasic:

```
Dim s As String
Dim a As Integer
Console.WriteLine(" Vnesi a:")
s = Console.ReadLine()
a=Integer.parseInt(s)
MsgBox("vrednost= " & a)
```

Matlab:

```
a=input(' vnesi a:');
disp([' vrednost= ',num2str(a)])
```

Kot lahko opazimo je Matlab edini, ki dovoljuje v enem stavku branje in pisanje na zaslon.

6.5 Vodenje toka programa

Algoritmi večinoma vključujejo odločitve, ki jih izvršimo s pogojnimi stavki, kot na primer: "*Če je vrednost X različna od nič izvedi deljenje z X*". To odločanje imenujemo razvejitev programa za katero uporabimo stavke **`IF`**. Druga oblika vodenja toka programa je možnost ponavljanja določenega dela kode večkrat, oziroma, dokler ni izpolnjen določen pogoj. Ta postopek imenujemo zanka. Zanko lahko nadomestimo s pogojnim stavkom, vendar je oblika zanke bistveno bolj pregledna, kar je odločilno pri uporabi. Zanke so v različnih programskih jezikih zapisane sintaktično različno, poznamo jih več oblik (**`for`**, **`DO`**, **`while`**) in se različno izvajajo. V nadaljevanju je opisano delovanje pogojnih stavkov in zank.

6.5.1 Pogojni stavki IF (logični operatorji)

Naslov sam je zapisan v množini, kar daje vedeti, da ima pogojni stavek **IF** več oblik. Pogojni stavki so prisotni v vseh programskih jezikih in ni programskega jezika, ki ga ne bi imel. Pogojne stavke zelo pogosto uporabljamo tudi v vsakdanjem jeziku, kadar želimo, da se določene aktivnosti zgodijo pri določenih pogojih. Pogojni stavki v programskih jezikih so izredno podobni vsakdanjemu pogovoru, "Če ključ ni pravi ne odklepa vrat", bi bil zelo enostaven, malo bolj zanimiv je, "Imaš pet ključev, poskusi najprej s prvim, če ne odgovarja, poskusi z drugim, če tudi ta ni pravi poskusi s tretjim, in tako naprej do zadnjega". Seveda bi pravi programer poskusil še s kakšnim boljšim postopkom - algoritmom. "Če je ključ cilinderski, preglej koliko imaš cilinderskih ključev in poskusi le z njimi" in s tem pospešil iskanje pravega ključa.

Najenostavnejša oblika stavka IF ima strukturo:

IF → pogoj → instrukcije
in ga lahko predstavimo: če je pogoj izpolnjen izvedi stavek, če ne, se naj program nadaljuje na naslednjem stavku.

Pogojni stavki delujejo s pomočjo primerjalnih in logičnih operatorjev. Primerjalni operatorji omogočajo medsebojno primerjavo vrednosti nad katerimi izvajamo operacije. Rezultat primerjave je logična vrednost pravilno (*angl. true*) ali nepravilno (*angl. false*) in od vrednosti rezultata je odvisno ali se stavek izvede ali ne. S primerjalnimi operatorji se srečujemo tako v pogojnih stavkih kot zankah.

Tabela 5: Primerjalni in logični operatorji.

Primerjalni operatorji	Fortran	VisualBasic	Java	Matlab
večji	>	>	>	>
večji ali enak	>=	>=	>=	>=
manjši	<	<	<	<
manjši ali enak	<=	<=	<=	<=
enak	==	=	==	==
neenak	/=	<>	!=	~=
Logični operatorji	Fortran	VisualBasic	Java	Matlab
in	.AND.	And	&	&
ali	.OR.	Or	|	|
negacija	.NOT.	Not	!	~

V večini primerov so enostavni pogoji redki. Sestavljanje logičnih operatorjev je zelo pogosto, saj so pogoji, ki jih želimo oblikovati večinoma kompleksnejši. Pri sestavljanju pogojev moramo paziti na prednostne operacije, ki delujejo pri enakovrednih operacijah od leve proti desni, večinoma pa je smotrno, da si pomagamo z oklepaji.

Velikokrat je potrebno ob določenem pogoju izvesti več aktivnosti, ali pa je pogojev več. Takrat uporabimo pogojni stavek v obliki bloka ukazov, ki ponazarjajo aktivnosti.

Oblike stavka IF

Enostavni *enovrstični:*

Fortran	`IF (n==0) X=X/100`
Java	`if (n==0) {X=X/100}`
VisualBasic	`if n=0 then X=X/100`
Matlab	`if n==0, then X=X/100`

Kot vidimo iz primerov se zapis sintaktično v različnih programskih jezikih razlikujejo, njihova funkcionalnost pa je popolnoma enaka.

Če je pri enakem pogoju potrebnih več aktivnosti **`IF`** zapišemo v obliki bloka stavkov. V programskem jeziku mora biti natančno izkazano, kje se stavek **`IF`** prične in kje se konča, zato je začetek bloka stavkov označen s stavkom **`IF`**, konec bloka pa enolično definiran s stavkom **`endif`, `end`** ali zaklepajem, odvisno od sintakse programskega jezika.

Fortran	***Java***	***VisualBasic***	***Matlab***
`IF (pogoj) THEN`	`IF (pogoj) {`	`IF pogoj Then`	***`if pogoj`***
`  stavki`	`  stavki`	`  stavki`	`  stavki`
`ENDIF`	`}`	`End If`	***`end`***

V nadaljevanju so podani primeri stavkov **`IF`** skupaj s pripadajočim diagramom poteka in kodo zapisano v različnih programskih jezikih.

Oblika *if then*

Primer: če je študent dosegel 60% točk ali več je izpit opravil; hkrati naj program sešteva število študentov s pozitivno oceno.

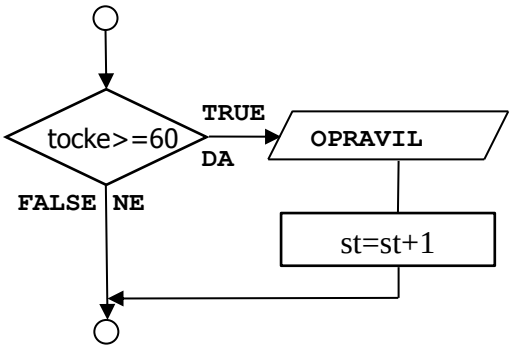
koda Fortran
```
IF(tocke>=60)THEN
  PRINT *,' OPRAVIL'
  ST=ST+1
ENDIF
```

koda Java
```
if (tocke>=60) {
  System.out.println(" OPRAVIL");
  st=st+1
}
```

koda VisualBasic
```
if tocke>=60 Then
  Debug.WriteLine(" OPRAVIL")
  st=st+1
End If
```

koda Matlab
```
if tocke>=60
  disp(' OPRAVIL')
  st=st+1;
end
```

Diagram poteka:

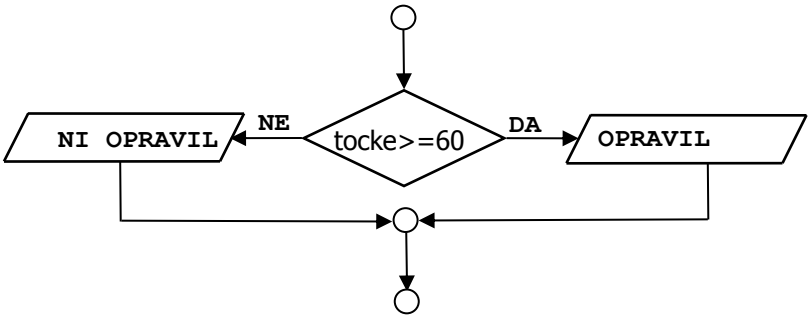


Oblika *if else*

omogoča izvajanje različnih aktivnosti če je pogoj izpolnjen ali če ni izpolnjen. V pogovornem jeziku bi rekli: Če je pogoj izpolnjen izvedi eno aktivnost, če pa ni, pa drugo.

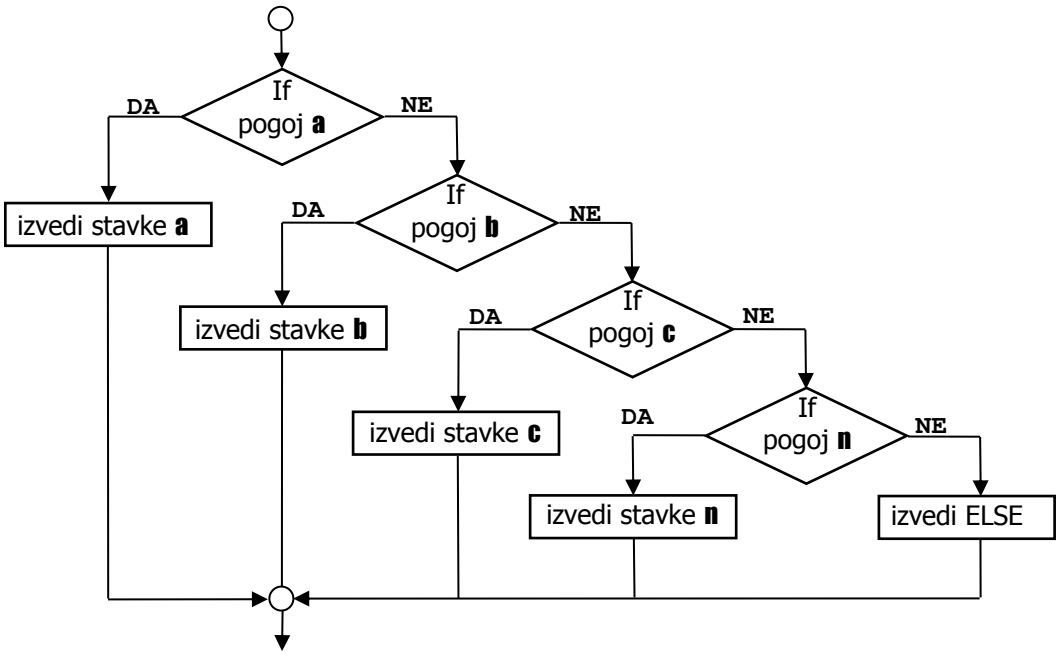
Primer: če je študent dosegel 60% točk ali več je izpit opravil, drugače ga ni opravil!

Diagram poteka:



koda Fortran
```
IF(tocke>=60)THEN
  PRINT *,'OPRAVIL'
ELSE
  PRINT*,'NI OPRAVIL'
ENDIF
```
koda Java
```
If (tocke>=60) {
  System.out.println("OPRAVIL");
} else {
  System.out.println("NI OPRAVIL");
}
```
koda VisualBasic
```
If tocke>=60 Then
  Debug.WriteLine("OPRAVIL")
Else
  Debug.WriteLine("NI OPRAVIL")
End If
```
koda Matlab
```
if tocke>=60
  disp('OPRAVIL');
else
  disp('NI OPRAVIL');
end
```

Stavki **`IF`** so lahko tudi znotraj drug drugega (so gnezdeni), poudariti pa je potrebno, da morajo biti v tem primeru znotraj drugega v celoti in da pri več pogojih postanejo nepregledni. To nepreglednost rešujemo s stavkoma **`If-elseif`** ali s stavkom **`Case`**, ki tudi delujeta hitreje. V nadaljevanju je na sliki prikazana diagram poteka **`If-elsif`** strukture pri pogojih in **`Case`** struktura. Najlaže pa je delovanje ponazoriti na primeru.



Slika 19: Diagram poteka If-elseIf.

Najlaže predstavimo delovanje na primeru. Če razširimo prejšnji primer točk pri izpitu z določitvijo ocene glede na doseženo število točk, lahko predpostavimo naslednja merila za oceno:

90 ali več: ODLIČNO
med 80 in 89: ZELO DOBRO
med 70 in 79: DOBRO
med 60 in 69: ZADOSTNO
pod 60: NI OPRAVIL
Predstavljena sta dva načina izvedbe: z zaporednimi stavki **If** in uporabi **If-elseIf** strukture.

Zaporedni stavki If

koda Fortran

```
IF (tocke >= 90) THEN
 Print*,"ODLIČNO"
ENDIF
IF(tocke>=80.AND.tocke<90)THEN
 Print*,"ZELO DOBRO"
ENDIF
IF(tocke>=70.AND.tocke<80)THEN
 Print*,"DOBRO"
ENDIF
IF(tocke>=60.AND.tocke<70)THEN
 Print*,"ZADOSTNO"
ENDIF
IF(tocke<60)THEN
 Print*,"NI OPRAVIL"
ENDIF
```

koda VisualBasic

```
If tocke >= 90 Then
 Debug.WriteLine("ODLIČNO")
End If
If tocke >= 80 And tocke < 89 Then
 Debug.WriteLine("ZELO DOBRO")
End If
If tocke >= 70 And tocke < 79 Then
 Debug.WriteLine("DOBRO")
End If
If tocke >= 60 And tocke < 69 Then
 Debug.WriteLine("ZADOSTNO")
End If
If tocke < 60 Then
 Debug.WriteLine("NI OPRAVIL")
End If
```

koda Java

```
if (tocke>=90) {
System.out.println("ODLIČNO");
}
if (tocke>=80 & tocke<90) {
System.out.println("ZELO DOBRO");
}
if (tocke >= 70 & tocke<80) {
System.out.println("DOBRO");
}
if (tocke >= 60 & tocke<70) {
System.out.println("ZADOSTNO");
}
if { tocke < 60) {
System.out.println("NI OPRAVIL");
}
```

koda Matlab

```
if  tocke >= 90
 disp ('ODLIČNO');
end
if (tocke>=80) & (tocke<90)
disp ('ZELO DOBRO');
end
if (tocke>=70) & (tocke<80)
 disp ('DOBRO');
end
if (tocke>=60) & (tocke<70)
 disp ('ZADOSTNO');
end
if tocke<60
 disp ('NI OPRAVIL');
end
```

If-ElseIf struktura

koda Fortran

```
IF (tocke >= 90) THEN
 Print*,"ODLIČNO"
ELSEIF(tocke>=80)THEN
 Print*,"ZELO DOBRO"
ELSEIF(tocke>=70)THEN
 Print*,"DOBRO"
ELSEIF(tocke>=60)THEN
 Print*,"ZADOSTNO"
ELSE
 Print*,"NI OPRAVIL"
ENDIF
```

koda VisualBasic

```
If tocke >= 90 Then
 Debug.WriteLine("ODLIČNO")
ElseIf tocke >= 80 Then
 Debug.WriteLine("ZELO DOBRO")
ElseIf tocke >= 70 Then
 Debug.WriteLine("DOBRO")
ElseIf tocke >= 60 Then
 Debug.WriteLine("ZADOSTNO")
Else
 Debug.WriteLine("NI OPRAVIL")
End If
```

koda Java

```
if (tocke >= 90) {
 System.out.println("ODLIČNO");
} else if (tocke >= 80) {
 System.out.println("ZELO DOBRO");
} else if (tocke >= 70) {
 System.out.println("DOBRO");
} else if (tocke >= 60) {
 System.out.println("ZADOSTNO");
} else {
 System.out.println("NI OPRAVIL");
}
```

koda Matlab

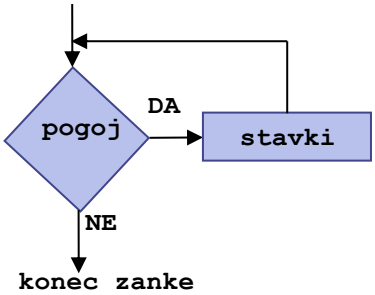
```
if  tocke >= 90
 disp ('ODLIČNO');
elseif (tocke>=80)
 disp ('ZELO DOBRO');
elseif (tocke>=70)
 disp ('DOBRO');
elseif (tocke>=60)
 disp ('ZADOSTNO');
else tocke<60
 disp ('NI OPRAVIL');
end
```

Vse zapisane variante kodiranja dajo enak rezultat:

1. Prva varianta je zapisana s pomočjo več zaporednih stavkov ***If***. Slabosti te kode v vseh programskih jezikih je predvsem nepreglednost; vsak stavek ***If*** deluje samostojno zaporedno eden za drugim, zato program preverja vsak stavek ***If*** in nujno je zapisati zgornji in spodnji pogoj; v primeru, da je *vrednost* spremenljivke *tocke* enak vrednosti 88, program pri izvajanju pri drugem stavku ***If*** izpiše ZELO DOBRO, nakar preverja vse naslednje stavke ***If***, od katerih pa nobeden ni aktiviran.
2. ***If-ElseIf*** oblika je bistveno bolj pregledna; omogoča kratko, jasno in elegantno rešitev, ki je najhitrejša tudi pri izvajanju kode.

6.5.2 Zanke

Zanke so eden najpogosteje uporabljenih programskih stavkov saj nam omogočajo, da del programske kode izvedemo večkrat. Število ponovitev nadziramo s pogojem (vsaj enim) ali štetjem (ki pa je prav tako pogoj). Telo zanke tvorijo vsi stavki, ki se ponavljajo, kar je v različnih programskih jezikih sintaktično različno zapisano. Fortran, VisualBasic in Matlab označujejo začetek in konec zanke s stavki, Java pa ima zapisano telo zanke znotraj zavitega oklepaja. Zanka se izvaja dokler je pogoj izpolnjen (logična zanka) ali po predpisanem številu ponovitev (številna zanka). Številne zanke dejansko izvedemo s pomočjo logičnih, kar je enostavno vidno iz shematskega prikaza delovanja zanke – diagrama poteka zanke.



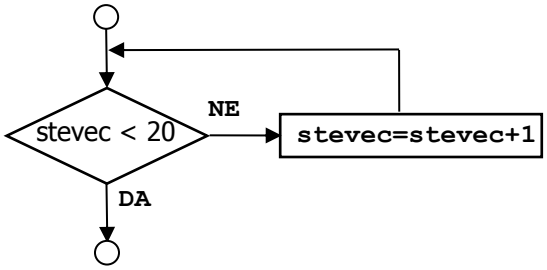
1. preverimo ali je pogoj izpolnjen,
2. če je pogoj izpolnjen, izvedemo stavke v zanki,
3. ponovno preverimo pogoj,
4. če je pogoj izpolnjen ponovno izvedemo stavke v zanki
5. ponavljanje dokler je pogoj izpolnjen
6. ponavljanje se konča ko pogoj ni več izpolnjen.

Logična zanka while

Logična zanka ima ob začetku, ki je oznaka za začetek zanke, zapisan pogoj, pri katerem se zanka izvede. Torej se izvede vsakič, ko je pogoj izpolnjen in se neha izvajati ko pogoj ni več izpolnjen. Če pogoj ni nikoli izpolnjen se zanka nikoli ne izvede, analogno pa velja, da morajo stavki v zanki vplivati na pogoj, drugače bo zanka delovala neskončno krat (ne preneha z delovanjem). Pri zankah moramo biti torej previdni, da je pogoj in izvajanje zanke usklajeno.

Logične zanke imajo zapisan pogoj v obliki pogojnega stavka, če je pogoj izpolnjen (*true*) se zanka izvede. Ta pogojni stavek je lahko zapisan na začetku ali koncu zanke, odvisno od namena in programskega jezika. Če pogoj zapišemo na koncu zanke, se zanka izvede vsaj enkrat. Fortran in Matlab nimata možnosti pogojnega stavka na koncu zanke.

diagram poteka in koda: *DO While* zanka



Zapisani so primeri zank logičnih zank, ki izpisujejo števila od 0 pa do števila določenega v pogoju (manjši od 20).

koda Fortran
```
stevec=0
DO WHILE (stevec < 20)
  stevec=stevec+1
ENDDO
```

koda Java
```
stevec=0;
while (stevec < 20) {
  stevec++
}
```

koda VisualBasic
```
stevec=0;
Do while stevec < 20
  Stevec+=1
Loop
```

koda Matlab
```
stevec=0;
while stevec < 20
  stevec=stevec+1;
end
```

V nadaljevanju je zapisanih nekaj možnosti izvajanja logičnih zank, za Fortran in Matlab na začetku zanke, za Java in VisualBasic pa obe varianti. Mogoče je potrebno še enkrat poudariti, da če je pogojni stavek na koncu zanke, se zanka izvede vsaj enkrat.

koda Fortran preverjanje na začetku zanke
```
INTEGER::x=0
DO WHILE (x<10)   ! preverja pogoj
  PRINT*,'x manjši od 10: x=',x
  x=x+1           ! povečanje x za 1
END DO
```

koda Matlab preverjanje na začetku zanke
```
x=0;
while x<10      % preverja pogoj
  disp (['x manjši od 10: x=',num2str(x)])
  x=x+1;        % povečanje x za 1
end
```

koda Java preverjanje na začetku zanke:
```
int x=0;
while (x<10)  // preverja pogoj
{
  System.out.println("x manjši od 10: x=" +
x);
  x++;        // povečanje x za 1
}
```

koda Java preverjanje na koncu zanke:
```
int x=0;
do
{
  System.out.println("x manjši od 10: x=" +
x);
  x++;  // povečanje x za 1
}
while (x<10);  // preverja pogoj
```

koda VisualBasic preverjanje na začetku zanke:

```
Dim x As Integer = 0
Do While x<10 ' preverja pogoj
  Debug.WriteLine("x manjši od 10: x=" & x)
  x+=1        ' povečanje x za 1
Loop
```

koda VisualBasic preverjanje na koncu zanke:

```
Dim x As Integer = 0
Do
  Debug.WriteLine("x manjši od 10: x=" & x)
  x+=1        ' povečanje x za 1
While x<10    ' preverja pogoj
Loop
```

Pogojni stavki v predstavljenih primerih zank so enostavni, seveda pa lahko zapišemo tudi kombinirane in kompleksnejše pogojne stavke:

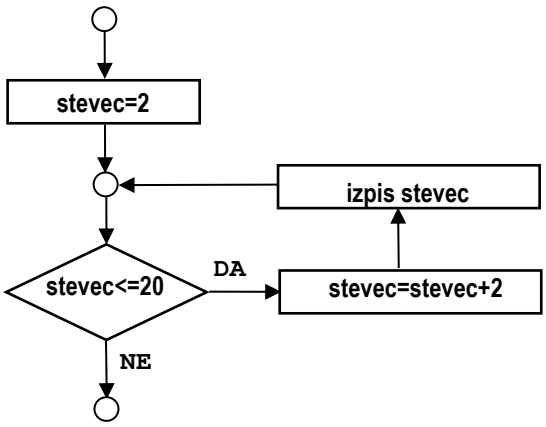
```
koda Fortran          DO WHILE ((x<10 .AND. x>1) .OR. c == 'YES')
koda Java             while (x<10 And x>1 Or c=='YES')
koda VisualBasic      Do While (x<10 And x>1) or c= YES
koda Matlab           while x<10 & x>1 | c=='Yes'
```

Zanka se bo izvajala ob pogoju da je vrednost 1<x<10, ali je v spremenljivki c znakovni niz Yes.

Številne zanke

Številna zanka je zapisana s pomočjo števca ali indeksa zanke, ki je praviloma celo število, nekateri programski jeziki pa dovoljujejo tudi uporabo realnega indeksa (Fortran, Matlab). V zanki mora biti definiran indeks zanke in njegov obseg, torej začetna in končna vrednost ter korak s katerim se naj vrednost indeksa spreminja. Končna vrednost indeksa v zanki ni nujno enaka končni vrednosti, saj ni nujno, da je sinhronizirana s korakom. Zanka se izvaja dokler je vrednost indeksa manjša ali enaka od predpisane končne vrednosti. Ni potrebno posebej poudariti, da lahko zanke tečejo v pozitivno ali negativno smer, le da so pogoji pri definiciji zanke pravilno zapisani, saj se izvede preverjanje indeksa vedno pred izvajanjem zanke. Indeks, začetna, končna vrednost in korak so lahko zapisani tako s konstantami ali spremenljivkami ali izrazi. V večini programskih jezikov koraka zanke ni potrebni pisati če ta znaša 1. V programerskih krogih velja, da indeksa znotraj zanke ne spreminjamo (Fortran to celo onemogoča), saj je takšno kodo izredno težko brati in popravljati.

Diagram poteka in koda v programskih jezikih za številno zanko, pri kateri teče števec od 2 do 10 s korakom 2 in izpisuje vrednost števca.



koda Fortran
```
DO stevec= 2,10,2
  Print*, stevec
ENDDO
```
koda Java
```
for (stevec=2; 10; stevec=stevec+2)
{
  System.out.println(stevec);
}
```
koda VisualBasic
```
For stevec = 2 To 10 Step 2
  Debug.WriteLine(stevec)
Next stevec
```
koda Matlab
```
for stevec = 2:2:10
  disp(stevec)
end
```

Večkratne (gnezdene) zanke

Prav tako kot pogojne stavke je mogoče tudi zanke gnezditi, kar pomeni, da lahko imamo znotraj ene zanke več drugih, govorimo o večkratnih zankah. V večkratnih zankah se je potrebno dosledno držati določenih navodil, ki jih določa sintaksa in semantika programskih jezikov.

1. če je ena zanka znotraj druge zanke pomeni, da mora biti znotraj druge v celoti,
2. večkratne zanke morajo imeti različne indekse.

Pri delovanju večkratnih zank je potrebno povedati tudi tole. Ko program zaključi izvajanje notranje zanke nadaljuje v zunanji zanki, ko pride ponovno do notranje zanke jo začne izvajati še enkrat od začetka. Delovanje je najlaže ponazoriti s števcem razdalje, na primer v avtomobilu ali kolesu. Tako lahko zapišemo program, ki bo samo štel metre in kilometre in ju sproti izpisoval do razdalje `n` recimo do 5 enot. To je najbolj tipičen in zelo nazoren primer uporabe dvojne zanke in zaradi enostavnosti so uporabljena cela števila.

Primer je zapisan na dva načina v programskem jeziku Fortran, saj je namen le ponazoriti delovanje. Najprej je koda zapisana kot jo večinoma zapišejo začetniki, nato pa kot programerji, z namenom predstaviti različnost kodiranja, ki je posledica znanja in izkušenj. Začetniki takšen del programa zapišejo v takšni obliki (razen mogoče stavka INTEGER, ki definira tudi začetne vrednosti, in imen zank, ki sta zapisani zaradi preglednosti):

```
INTEGER:: i,j,m=0,km=0,n=5
zunanja: DO i=1,n
  notranja: DO j=1,1000
    m=m+1
    PRINT *,' metri=',m
  ENDDO notranja
  km=km+1
  PRINT *,' kilometri=',km
  m=0
ENDDO zunanja
PRINT*,'i=,i,' j=',j
```

Notranja zanka uporablja indeks **j** za štetje metrov zunanja pa indeks **i** za štetje kilometrov. Zunanja postavi indeks **i=1**, nato pa izvajanje prevzame notranja zanka z indeksom **j**. Notranja zanka se izvede 1000 krat (od 1 do 1000 s korakom po ena), v kateri se število metrov vsakič poveča za ena in izpiše. Notranja zanka preneha z izvajanjem pri indeksu **j=1001**, ki je večji od končne vrednosti in program nadaljuje izvajanje na naslednjem stavku po zaključku zanke. Ker je bilo preštetih 1000m se poveča kilometer za **1** in se števec metrov **j** zopet postavi na ena za začetek novega kilometra. Iz primera je razvidno, da se števci zank uporabljajo le za štetje števila ponovitev. Zadnji stavek izpiše vrednosti indeksov po izvajanju zank, ki znašata (**i=6** in **j=1001**), kar nazorno prikazuje izvajanje zank, ker se indeksi povečujejo dokler ne presežejo končne vrednosti zanke.

```
INTEGER:: m,km,n=5
zunanja: DO km=1,n
  notranja: DO m=1,1000
    PRINT *,' metri=',m
  ENDDO notranja
  PRINT *,' kilometri=',km
ENDDO zunanja
```

Drugi primer programa uporablja za štetje kar indekse in zato potrebuje manj spremenljivk in stavkov. Ker je indeks za metre vsakič pri začetku izvajanja zanke definiran z začetno vrednostjo ni potrebno postavljati števca metrov na začetek, kot pri prejšnjem primeru. Zunanja zanka postavi števec km na 1 (začetek prvega kilometra) in nato šteje metre od 1 do 1000, in jih sproti izpisuje. Ko notranja zanka konča izvajanje (vrednost **m=1001**), se izvajanje nadaljuje na naslednjem stavku, ki izpiše kilometre in nadaljuje izvajanje na zunanji zanki, indeks **km=2** (začetek drugega kilometra) in nato sledi izvajanje notranje zanke (1000 krat) za naslednji kilometer in tako naprej, dokler ni zaključena tudi zunanja zanka. Ni potrebno posebej poudariti kateri zapis je hitrejši pri izvajanju, bolj pregleden in učinkovit.

Sama struktura in logično izvajanje zank definira nekatere omejitve, ki jih imamo pri sestavljanju zank:

- Preskok iz drugega dela programa zanko ni možen.
- Pri gnezdenih zankah ali sestavljenih strukturah DO, CASE in IF če je ena znotraj druge mora biti znotraj v celoti.
- Gnezdene zanke morajo imeti različne indekse.
- Indeksov in v številnih zankah parametrov zanke znotraj ne spreminjamo.

V telo zanke iz ostalega dela programa ne moremo, zanka se začne s prvim stavkom zanke in konča z zadnjim. Programski jeziki pa nam omogočajo predčasni izhod iz zanke ali preskok na naslednji indeks zanke. V zankah možnost prekiniti izvajanje aktivne zanke (izhod iz zanke) ali preskakovati na konec aktivne zanke. Uporabnost najlaže ponazorimo s primerom.

Predpostavimo, da preiskujemo seznam študentov, ki vključuje na primer: ime, priimek, številko indeksa, leto vpisa in število opravljenih izpitov. V zanki preiskujemo celotni nabor študentov. Predčasni izhod iz zanke je uporaben, če na primer iščemo študenta z določeno številko indeksa in ko jo najdemo, uporabimo podatke in ne želimo nadaljevati preiskovanja seznama še naprej, ker bi bilo to popolnoma nepotrebno. Pri izhodu iz zanke, program nadaljuje izvajanje na prvem naslednjem stavku za koncem aktivne zanke (lahko na primer v drugi zanki, če sta gnezdeni). Preskok bi uporabili kadar zbiramo podatke o študentih, ki so se vpisali določenega leta. Ko

preberemo podatke za posameznega študenta ugotovimo, da študent ni bil vpisan določenega leta, torej nam podatki za delo niso zanimivi za delo in enostavno preskočimo na naslednji indeks zanke, torej na zapis za naslednjega študenta.

Kot primer je zapisan del programa s trojno zanko, ki samo izpisuje indekse zank, tako, da je mogoče enostavno slediti izvajanju.

Tabela 6: Potek vrednosti indeksov pri trojni zanki.

število ponovitev	indeks *i*	indeks *j*	indeks *k*
1	1	1	-2
2	1	1	-3
3	1	2	-2
4	1	2	-3
5	2	1	-2
6	2	1	-3
7	2	2	-2
8	2	2	-3
9	3	1	-2
10	3	1	-3
11	3	2	-2
12	3	2	-3

koda Matlab

```
n=3;
for i=1:n
  for j=1:n-1
    for k=n-5:-1:-3
      fprintf(' indeksi %d %d %d',i,j,k)
    end
  end
end
```

koda Fortran

```
INTEGER::i,j,k,n=3
DO i=1,n
  DO j=1,n-1
    DO k=n-5,-3,-1
      PRINT*,'indeksi ',i,j,k
    ENDDO
  ENDDO
ENDDO
```

koda Java

```
int n=3;
for(int j=1; i<n; i++)
{
  for(int j=1; j<n-1; j++)
  {
    for (int k=n-5; k<-3; k--)
    {
      System.out.println("indeksi " +i+j+k);
    }
  }
}
```

koda VisualBasic

```
Dim n As Integer = 3
For i As Integer = 1 To n
    For j As Integer = 1 To n-1
        For k As Integer = n-5 To -3 Step -1
            Debug.WriteLine("indeksi " & i & j & k)
        Next k
    Next j
Next i
```

Vsi programski jeziki imajo možnost izhoda iz zanke, nimajo pa vsi možnosti preskoka na konec trenutno aktivne zanke (na primer VisualBasic). Fortran ima dodatno možnost, če so zanke označene z labelami, se pri izhodu iz zanke sklicujemo na ime zanke in lahko preskočimo iz več zank.

jezik	izhod iz aktivne zanke	preskok na konec aktivne zanke
Fortran	**EXIT**	**CYCLE**
Java	**break**	**continue**
VisualBasic	**Exit For, Exit Do**	
Matlab	**break**	**continue**

Najlaže je delovanje zank ponazoriti s primeri. V zankah zelo pogosto uporabljamo tudi pogojne stavke ali bloke. Osnovno pravilo zank ostaja enako: če je en blok znotraj drugega mora biti znotraj celotni blok. V nadaljevanju je zapisanih nekaj zank v različnih programskih jezikih, ki ponazarjajo delovanje zank in možnosti izhoda iz zank.

koda Fortran

```
REAL :: ocena, poprecje
INTEGER:: st, zanka
glavna_zanka: DO
  PRINT*,'Vnesi stevilko studenta'
  READ*,st
  IF(st==0) EXIT glavna_zanka
  poprecje=0
  ocene_zanka: DO zanka=1,3
    PRINT*,'vnesi oceno'
    READ*,ocena
    IF(ocena==0)CYCLE ocene_zanka
      IF(ocena<0)THEN
        PRINT*,'napačna ocena!
Ponovi!'
        CYCLE glavna_zanka
      ENDIF
    poprecje=poprecje+ocena
  ENDDO ocene_zanka
poprecje=poprecje/3
PRINT *,'poprecje studenta',st,' = ',
poprecje
ENDDO glavna_zanka
```

koda Java

```
for (int x = 0; x < 5; x++)    //zunanja zanka
{
   System.out.println("Zunanja: " + x);
   for (int y = 0; y < 5; y++) //notranja zanka - continue gre sem
   {
      System.out.println("Notranja: " + y);
      if (x > 2) {continue;}   //gre na začetek notranje zanke
      if (x > 3) {break;}      //izhod iz notranje zanke
   }
   //break gre sem, ostane v zunanji zanki
}
```

6.5.3 Indeksirane spremenljivke in polja

Podatke v programu shranjujemo v spremenljivke. Vsaka spremenljivka ima svoje ime. Iz zapisa in dodeljevanja vrednosti spremenljivki je povsem jasno in logično, da je lahko v eni spremenljivki shranjena le ena sama vrednost. Če vrednost spremenljivke s prireditvenim stavkom spremenimo je nova vrednost prepisana preko stare in je stara vrednost izgubljena. Večkrat bi bilo bolj prikladno, da bi določeno skupino podatkov shranili skupaj v podatkovno strukturo, kot pa da vsak podatek hranimo v posamezni spremenljivki. Najbolj pogosta uporaba podatkovnih struktur je polje. Da bi se izognili izredno nepreglednemu načinu dela, so bile v računalniških programskih jezikih zelo zgodaj uvedena polja ali kakor jih tudi imenujemo indeksirane spremenljivke. Polja so strukture predpisane velikosti, kje shranimo več vrednosti enakega tipa (na primer celoštevilčne ali znakovne nize). Podatki so v strukturi shranjeni z istim imenom, ločijo pa se po indeksu. Polja imajo dimenzionalnost (število indeksov) in velikost (obseg posameznega indeksa). Tako lahko imamo enodimenzionalna polja ali večdimenzionalna polja. Indeksi so praviloma lahko le cela števila.

Dimenziji moramo indeks definirati s spodnjo in zgornjo mejno vrednostjo, spodnjo lahko izpustimo, kar pomeni da je spodnja v Fortran in Matlab enaka 1 v ostalih programskih jezikih pa 0. V vseh programskih jezikih, razen Matlab, je potrebno polje deklarirati, predpisati tip in velikost in razsežnost polja. Programski jeziki imajo različne rešitve tudi za dinamično spreminjanje razsežnosti in dimenzije polj, razen Matlab, ki to počne avtomatsko. Posebnost Java je, da nima večdimenzionalnih polj. Polja moramo najprej deklarirati in nato kreirati, na voljo imamo le enodimenzionalno polje, vendar so lahko posamezne dimenzije zopet polja in tako preidemo do večdimenzionalnih polj, kot prikazuje spodnji primer.

Nasploh v matematiki, posledično tudi v numeriki in računalništvu, velikokrat uporabljamo indekse kadar govorimo o več vrednostih, ki imajo skupno ime, npr: koordinate x,y,z točk v prostoru ali matrike.
$\{x_i\}=\{x_1, x_2, x_3, x_4, x_5\}$; enodimenzionalno polje (imenujemo tudi vektor)

$\{a_{ij}\}=$ $\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \\ a_{m1} & & a_{mn} \end{pmatrix}$ dvodimenzionalno polje (matrika)

Najlaže ponazorimo uporabnost polj na enostavnem primeru seštevanja števil. Če predpostavimo, da bi želeli sešteti niz 10 števil, bi morali deklarirati in v programu uporabljati 10 spremenljivk in tudi programiranje (na primer izračun povprečne vrednosti niza števil) bi bilo zelo nečitljivo.

Branje števil: **`st1, st2, st3, st4, st5, st6, st7, st8, st9, st10`**

Izračun povprečja: **`pop=(st1+st2+st3+st4+st5+st6+st7+st8+st9+st10)/10`**

Takšen način dela je zelo, zelo slabo programiranje. Kako bi nalogo opravili, če bi morali prebrati niz 100 ali 9999999 števil ali v naprej nedefinirano število števil ?

Poglejmo si zasnovo programa za računanje poprečja niza 10 števil malo drugače. Predpostavimo, da imamo delno vsoto, imenujemo jo **`d_v`**, h kateri prištevamo posamezna števila. Tako v programu predvidimo, da k delni vsoti vedno znova prištevamo naslednje število iz niza. Kot vidimo iz dela programa v prvem stolpcu se seštevanje nadaljuje v desetih korakih. Da ni potrebno pisati vseh delnih vsot je uporabljen indeks. Indeksi in zanke pa tvorijo najmočnejše orodje pri programiranju. V drugem stolpcu je zapisano seštevanje desetih števil s pomočjo zanke v Fortran in Matlab. Indeksi se v vseh programskih jezikih pišejo v oklepaju.

```
d_v = 0.0
d_v = d_v + st1
d_v = d_v + st2
...
d_v = d_v + st"i"
...
d_v = d_v + st10
pop = d_v / 10
```

```
do i=1,10
  d_v = d_v + st(i)
enddo
pop = d_v / 10.0
```

```
for i=1:10
  d_v = d_v + st(i)
end
pop = d_v / 10
```

Seveda je v vsakem programskem jeziku interna funkcija, ki avtomatsko izračuna vsoto vrednosti v polju. Primer služi za razlago , kako zanke in polja delujejo. Če polja ponazorimo grafično lahko uporabimo naslednji opis.

spremenljivka A

	A	9.000000

polje b(0:4)

	b	0	-10.00
	b	1	0.00
	b	2	100.00
	b	3	50.00
	b	4	-50.00

polje c(1:2,1:2)

	c	1	1	-1.00
	c	2	1	0.00
	c	1	2	1.00
	c	2	2	0.00

polje F(2,3,2)

F	1	1	1	0
F	2	1	1	0
F	1	2	1	0
F	2	2	1	0
F	1	3	1	0
F	2	3	1	0
F	1	1	2	0
F	2	1	2	0
F	1	2	2	0
F	2	2	2	0
F	1	3	2	0
F	2	3	2	0

vrstni red

1
2
3
4
5
6
7
8
9
10
11
12

matematični zapis matrike in zapis v računalniškem pomnilniku

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

A(1,1)
A(2,1)
A(3,1)
A(1,2)
A(2,2)
A(3,2)
A(1,3)
A(2,3)
A(3,3)

Zelo nazorni primer dvodimenzionalnega polja je matrika c (tabela). Spremenljivka, ki opisuje matriko je je dvodimenzionalna, ima dva indeksa, s katerima lahko opišemo položaj vsakega člena v matriki prvi indeks je številka stolpca in drugi številka vrstice. Prioriteta indeksov v računalništvu je od levega k desnemu, kar ni enako zapisu, kot ga uporabljamo pri matematiki. Polje F prikazuje trodimenzionalno polje, katerega razsežnosti so 2x3x2, torej 12 vrednosti.

Polja so bila in so še vedno ena od najpomembnejših podatkovnih struktur v programskih jezikih. To je tudi vzrok za nenehen razvoj možnosti in sprememb pri deklaraciji in funkcij za delo s polji. V razvoju so bila najprej le deklaracije fiksne razsežnosti polj, kar je pomenilo, da razsežnosti polj med izvajanjem programa ne moremo spreminjati. Danes večina programskih jezikov omogoča dinamično dodeljevanje (*angl. allocate*) velikosti polj, ki omogoča med izvajanjem programa definiranje in spreminjanje velikosti polj. Sam postopek dinamičnega dodeljevanja je malo bolj kompleksen kot pri fiksnem dodeljevanju velikosti polj, vendar je vreden dodatnega dela. V nekaterih programskih jezikih kot je na primer Java pa so polja vedno dinamično deklarirana.

Indeksirane spremenljivke in zanke v kombinaciji dajejo izredno močno orodje za programiranje. Poudariti je še potrebno, da je lahko indeks število, ali celoštevilčni izraz.

```
A(i)=b(i)+c(i-1,i+2)
```

Deklaracijo in kreiranje polja lahko v Java izvedemo v enem koraku. Podan je primer uporabe definiranja polj v Java in Fortran, za primerjavo primernosti jezikov za numerične probleme. Programa sta napisana za ponazoritev delovanja, in seveda, kodirati bi jih bilo mogoče bolj učinkovito a manj nazorno. Na splošno lahko rečemo, da sta Fortran im Matlab prav na področju polj najučinkovitejša programska jezika. Fortran je polja uvedel v programske jezike, Matlab pa tako in tako temelji na poljih. Za ponazoritev je prikazan primer kode v Java in Fortran z enakim rezultatom, izpisom vrednosti matrike 5x4. Že sam pogled na kodo, brez predznanja o programiranju je dovolj zgovoren.

koda Java

```
public class JAVAPolja {
	public static void main(String[] args) {
		int[][] Matrika = new int[4][];
	// zapolnimo matriko z vrednostmi
		for (int i = 0; i <Matrika.length; i++) {
			Matrika[i] = new int[5];  //kreiranje (pod)polja
			for (int j = 0; j < Matrika[i].length; j++) {
				Matrika[i][j] = i + j;
			}
		}
	//izpis matrike
		for (int i = 0; i < Matrika.length; i++) {
			for (int j = 0; j < Matrika[i].length; j++) {
				System.out.print(Matrika[i][j] + " ");
			}
		System.out.println();
		}
	}
}
```

koda Fortran

```
Program FortranPolja
Integer:: Matrika(4,5), i, j    !deklaracija od 1:5
DO i=1,4
  DO j=1,5
    Matrika(i,j) = i + j ! zapolnimo matriko z vrednostmi
  ENDDO
ENDDO;
DO i=1,4
  DO j=1,5
    Print*,Matrika(i,j) ! izpis matrike
  ENDDO
ENDDO
```

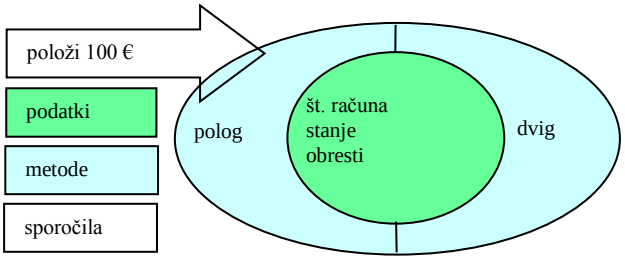
Rezultat izvajanja programa:

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
```

6.6 Strukturirano in objektno programiranje

Obsežni programi zahtevajo strukturirano programiranje. Osnova za razvoj je bilo ponavljanje določenega dela kode večkrat v programu. Tako so nastale funkcije in podprogrami. Funkcije in podprogami so zaključeni deli kode, ki lahko komunicirajo z drugimi deli preko atributov in parametrov. Zgoraj navedeni razlogi so bili temelj za nastanek strukturiranega in objektnega programiranja. Objektno programiranje se je izkazalo za koristno, če podatke in postopke s podatki združujemo v celoto – objekt. Programiranje je potem sestavljeno iz ustvarjanja objektov in povezovanja teh objektov v izvajanje postopkov. Večina sodobnih programskih jezikov je objektno usmerjenih. Ti programski jeziki uporabljajo aktivnosti kot: združevanje (*enkapsulacija*), večličnost (*polimorfizem*) in dedovanje (*inheritenca*). Objektno orientirano programiranje omogoča, da na problem gledamo kot na množico objektov, ki sodelujejo in vplivajo drug na drugega, kar potrebujemo pri programih, ki uporabljajo hkrati več zaslonskih funkcij, kjer se odvija večje število dogodkov hkrati, ki se med seboj prepletajo (miška, zapiranje in odpiranje oken, izbira in podobno). Takšne programe bi bilo s postopkovnim reševanjem

(nizanjem ukazov) izredno težko zapisati. Programski jeziki, kot so Java, C# nudijo okolja, v katerih so že določeni objekti, ki imajo lastnosti nad katerimi izvajamo metode in funkcije.



Objekt je samostojen programski modul, ki vsebuje:

podatke: opisujejo (notranje) stanje objekta

metode: podprogrami, ki delujejo nad podatki

Lastnosti objektov:

- povezanost podatkov in metod, ki po sodijo skupaj,
- dobro definirani in nadzorovani vmesniki do objekta,
- dedovanje: hierarhično izpeljevanje novih objektov iz obstoječih z dodajanjem podatkov in/ali spreminjanjem metod,
- enkapsulacija, "skrivanje" notranjih podatkov pred uporabo od zunaj in podrobnosti realizacije, ki omogočajo večjo prilagodljivost

Ko v takšnem okolju pišemo programe ni potrebno zapisati na primer kode za vnos datuma preko okna, ki vsebuje koledar, temveč, da v okolju že imamo takšen objekt, ki ga uporabimo s pripravljenimi metodami. Objektno usmerjeno programiranje pomeni, da na problem gledamo kot na niz objektov, ki med seboj sodelujejo in vplivajo drug na drugega. Večina objektnih jezikov ima v okolju že definirane objekte in metode za delo z njimi. Objekti so grafično okno, tiskalnik, slika na spletni strani, ki jim z metodami spremenimo lastnosti kot so velikost, ime datoteke za izpis in podobno.

Osnove objektnega programiranja

Metoda v Java ali VisualBasic je ločen blok kode, ki izvaja določene aktivnosti. S pomočjo metod razdelimo program na več manjših, enostavno razumljivih delov, ki tvorijo logične celote in jih lahko uporabimo večkrat.

Metode uporabljamo v programih zaradi čitljivosti, pa tudi kompleksnejše programe razdelimo v jasno strukturo. Kako razdeliti programe na metode je večinoma stvar izkušenj in možnosti, ki nam jih nudi programsko okolje. Vsak del kode, ki opravlja namensko funkcijo lahko zapišemo kot metodo. Metode so zaključene programske enote, ki jim večinoma posredujemo podatke, ki jih imenujemo parametri. Če v programu večkrat računamo ploščino kroga, je smiselno napisati metodo, ki iz podanega premera izračuna ploščino. Metoda v tem primeru potrebuje podatek kot vhodni parameter in vrne vrednost za ploščino kot izhod. Takšno metodo lahko uporabimo kolikokrat hočemo s pomočjo klicanja metode. Torej, vsakič ko metodo potrebujemo jo pokličemo in ji preko parametrov posredujemo podatke.

Objektno programiranje temelji na objektih in metodah. Objekti v programiranju so zaključene celote, za katere nam ni potrebno vedeti kako delujejo znotraj objekta, poznamo samo vhodne in izhodne parametre. To lastnost imenujemo enkapsulacija. S predmeti delamo preko metod in lastnosti. Metode so način, s katerim povemo predmetu, kaj naj dela; lastnosti pa opisujejo značilnosti predmeta. Torej metoda ni samo določen zaključen del programske kode, pač pa definicija kako se predmet vede. V objektnem programiranju večinoma uporabljamo izraz pozvati in ne poklicati.

Objektno orientirani programski jeziki omogočajo velikokratno uporabo kodiranih rešitev in prilagajanje že izvedene rešitve za nove potrebe. Na osnovi tega je prišlo do delitve med programerji: dobri programerji pišejo dobre programe, odlični programerji si “sposodijo” dobro kodo in jo modificirajo.

7 Literatura in viri

P.K. Sinha, P. Sinha; Computer Fundamentals; BPB Publications; 2004

L. Kostrevc; Računalništvo in informatika, četrta izdaja, Pasadena, 2006

B. Forouzan, F.Mosharraf; Foundations of computer science, 2nd edition; Thomson, 2007

G. Anderson, D. Ferro, R. Hilton; Connecting with Computer Science, 2nd edition; Cengage Learning, 2010

A Tanarenko; Osnove računalništva, FNM Univerza v Mariboru, 2011 (www.matematika-racunalnistvo.fnm.uni-mb.si)

T.C. Chiang; Introduction to Computer Science; (http://web.ntnu.edu.tw/~tcchiang) (dec.2012)

G.M. Schneider, J. Gersting; Invitation to Computer Science, 6th edition; Cengage Learning, 2013