

# Uvod u programiranje

Skipta za kurs

Uvod u programiranje kroz JavaScript

Una Stanković

unastankovic1310@gmail.com

10. septembar 2018.

U ovom tekstu predstavljene su teorijske osnove potrebne za savladavanje kursa " Uvod u programiranje kroz JavaScript". Najpre su navedene, ukratko, teorijske osnove računarstva i uvod kroz HTML i CSS, a kasnije se ulazi u rad sa JavaScript-om. Ova skripta je obavezan materijal pri kursu i sa prezentacijama i kodovima formira celinu. Ova skripta sama po sebi nije dovoljna, samostalan rad i istraživanje je neizostavni deo procesa učenja. U slučaju da primetite greške pri čitanju rada ili imate bilo kakve nedoumice, predloge i sugestije javite se mejlom na adresu navedenu na prvoj strani.

## Sadržaj

<b>1</b>	<b>Uvod u računarstvo</b>	<b>4</b>
1.1	Istorijat računarstva . . . . .	4
1.2	Fon Nojmanova arhitektura . . . . .	6
1.3	Hardver . . . . .	9
1.4	Softver . . . . .	9
1.5	Oblasti savremenog računarstva . . . . .	10
1.6	Osnovni pojmovi i konstrukti . . . . .	11
1.7	Klasifikacija programskih jezika . . . . .	15
1.8	Primeri za vežbu . . . . .	15
1.9	Domaći zadatak . . . . .	16
<b>2</b>	<b>Uvod u web</b>	<b>17</b>
2.1	Uloga računarskih mreža . . . . .	17
2.2	Komponente računarskih mreža . . . . .	18
2.2.1	Mrežni hardver . . . . .	18
2.2.2	Komunikacioni kanali . . . . .	19
2.2.3	Mrežni softver . . . . .	21
<b>3</b>	<b>HTML i CSS</b>	<b>22</b>
<b>4</b>	<b>Zaključak</b>	<b>23</b>
	<b>Literatura</b>	<b>23</b>
<b>A</b>	<b>Dodatak</b>	<b>23</b>

# 1 Uvod u računarstvo

Računarstvo i informatika predstavljaju jednu od najvažnijih oblasti današnjice koje su u konstantnom razvoju. Danas, ne možemo zamisliti život bez računara, pametnih telefona ili mnogobrojnih uređaja koji se pokreću uz pomoć računara. Razvitak računarstva i tehnologije u poslednjih 70 godina je eksponencijalan, tako da, danas, imamo razvoj prenosnih računara, tableta, pametnih telefona i uređaja, kola, kućnih aparata i ostalih koji se pokreću korišćenjem računarskih sistema. Kako definisati računarski sistem? Postoji više različitih računarskih sistema, od kojih svaki ima svoju posebnu definiciju, ali naš fokus je na digitalnim računarskim sistemima. Oni podrazumevaju mašinu koja može da se programira kako bi izvršavala različite zadatke svođenjem na elementarne operacije nad brojevima. Brojevi se u računaru zapisuju uz pomoć nula i jedinica, odnosno, binarnim zapisom, kao nizovi bitova.

Kada se razmišlja o tome šta sve računarstvo obuhvata, lako se uviđa da računarstvo nije samo računar, već da ono predstavlja mnogo širu oblast koja se bavi izučavanjem teorije i prakse procesa računanja i primene računara u različitim naučnim oblastima, tehnicima i svakodnevnom životu. Računar sam po sebi nije cilj, već sredstvo za postizanje različitih ciljeva u zavisnosti od njihove primene. Za današnje računare često ćemo čuti da su "programabilni", ali šta to zapravo govori? Programabilnost računara se ogleda u činjenici da je moguće računaru dati neki skup instrukcija koje će on izvršavati sa ciljem ispunjavanja određenih zadataka, koje mu čovek, odnosno, programer zadaje. Računari kakve danas poznajemo nastali su polovinom XX veka, ali želja za automatizacijom određenih postupaka seže daleko dalje u prošlost. Naime, posmatrajući istorijski, ljudi su vekovima stvarali razne naprave koje su mogle da rešavaju neke numeričke zadatke.

## 1.1 Istorijat računarstva

Da bismo u potpunosti razumeli računarstvo moramo imati uvid u njegove početke i razvoj. Istorijski gledano, koreni ljudske želje da olakšaju sebi svakodnevni život sežu davno u prošlost. Kao pravi primer takvih težnji možemo uzeti jednu od prvih računaljki abakus. U 18. veku nastale su prve mehaničke sprave koje su mogle da vrše automatsko izvođenje aritmetičkih operacija i pomažu u rešavanju matematičkih zadataka. Blez Paksal<sup>1</sup> je 1642. godine konstruisao mehaničke sprave koje su služile za sabiranje i oduzimanje celih brojeva, zvane Paskaline. Trideset godina nakon njega, Godfrid Lajbnić<sup>2</sup> konstruisao je mašinu, zasnovanu na dekadnom sistemu, koja je mogla da vrši sve četiri osnovne operacije. Lajbnić je bio prvi koji je predlagao koriscenje binarnog sistema.

**Mehaničke mašine** Žozef Mari Žakard<sup>3</sup> je 1801. godine napravio prvu programabilnu mašinu — mehanički tkački razboj. On je pomoću bušenih kartica kreirao kompleksne šare na tkanini. Svaka rupa na kartici određivala je

---

<sup>1</sup>Blaise Pascal (1623–1662), francuski filozof, matematičar i fizičar. U njegovu čast jedan programski jezik nosi ime PASCAL.

<sup>2</sup>Gottfried Wilhelm Leibniz (1646–1716), nemački filozof i matematičar

<sup>3</sup>Joseph Marie Jacquard (1752–1834), francuski trgovac.

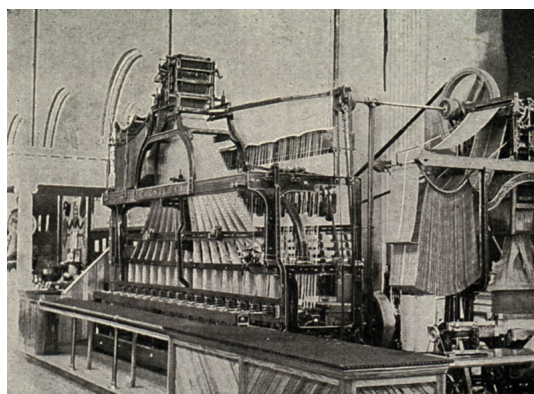


Slika 1: Abakus



Slika 2: Paskalina

jedan pokret mašine, a svaki red na kartici odgovarao je jednom redu šare. U

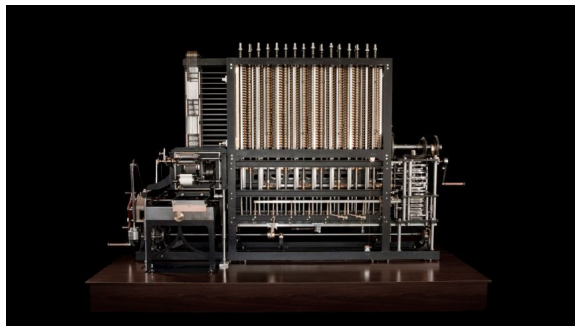


Slika 3: Žakardov razboj

prvoj polovini 19. veka, Čarls Bebidž<sup>4</sup> je dizajnirao prve programabilne računske mašine. Godine 1822. započeo je rad na diferencijskoj mašini, za računanje vrednosti polinomijalnih funkcija. Ime je dobila zbog toga što je koristila tzv.

<sup>4</sup>Charles Babbage (1791–1871), engleski matematičar, filozof i pronalazač.

metod konačnih razlika da bi bila eliminisana potreba za množenjem i deljenjem. Mašina je trebalo da ima oko 25000 delova i da se pokreće ručno, ali nije nikada završena. Ubrzo nakon toga, Bebidž je započeo rad na novoj mašini nazvanoj analitička mašina. Osnovna razlika u odnosu na sve prethodne mašine specifičnih namena, bila je u tome što je analitička mašina zamišljena kao računska mašina opšte namene. "Programiranje" na ovoj mašini vršilo bi se programima zapisanim na bušenim karticama (sličnim Žakardovim), a program bi kontrolisao mehanički računar koji bi omogućavao sekvencijalno izvršavanje naredbi, grananje i skokove. Osnovni delovi ovog računara trebalo je da budu mlin (engl. mill) i skladište (engl. store), koji po svojoj funkcionalnosti sasvim odgovaraju procesoru i memoriji današnjih računara. Ada Bajron<sup>5</sup> zajedno sa Bebidžem napisala je prve programe za analitičku mašinu i, da je mašina uspešno konstruisana, njeni programi bi mogli da računaju određene složene nizove brojeva (takozvane Bernulijeve brojeve). Upravo je to razlog zašto se ona smatra prvim programerom u istoriji. Ona je bila i prva koja je uvidela da se računarske mašine mogu upotrebiti i u nematematičke namene, čime je naslutila današnjicu.



Slika 4: Bebidžova diferencijska mašina

**Elektromehaničke mašine** Ove mašine koristile su se od sredine 19. veka do Drugog svetskog rata. Jednu od prvih mašina za čitanje bušenih kartica konstruisana je od strane Hermana Holerita. Njena glavna svrha bila je obrada rezultata popisa stanovništva u Sjedinjenim američkim državama 1890. godine. Koristeći bušene kartice uspešno izvršen je popis za godinu dana, naspram deset godina, koliko je bilo potrebno ranije. Od Holeritove male kompanije nastao je IBM.<sup>6</sup>

**Elektronski računari** Elektronski računari koriste se od kraja 1930-ih do danas.

## 1.2 Fon Nojmanova arhitektura

Na osnovu istorijata može se uočiti da svi navedeni računari imaju nedostatak jedne važne karakteristike računara danas, a to je programabilnost. Mašine

<sup>5</sup>Augusta Ada King (rođ. Byron), Countess of Lovelace, (1815–1852), engleska matematičarka. U njenu čast nazvan je programski jezik ADA.

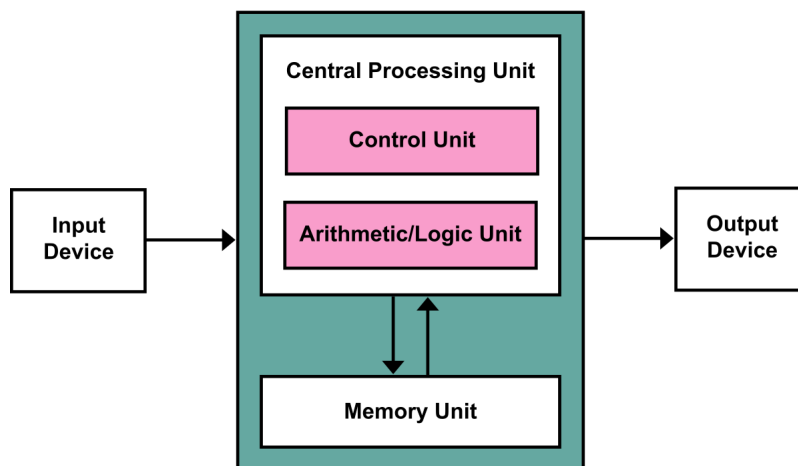
<sup>6</sup>Herman Hollerith (1860–1929), američki pronalazač.

korišćene nekada nisu bile programabilne već su funkcionisale po unapred definisanom programu određenom samom konstrukcijom mašine. Iako ovakav pristup nije sasvim izumro (danas se može videti na primeru digitrona), uglavnom nije poželjan. Prava promena u pristupu, koja je dovela do stvaranja programabilnih računara, nastala je ranih 1940-ih godina sa pojavom računara koji bi programe koje izvršavaju čuvali u memoriji zajedno sa podacima. Takve računare nazivamo računarima sa skladištenim podacima (engl. stored program computers). Jedna od najvažnijih karakteristika ovih računara je da kod njih postoji jasna podela na hardver i softver. Za rodonačelnika ovakve arhitekture smatra se Džon fon Nojman. On je 1945. godine opisao arhitekturu čija je glavna karakteristika da se programi mogu učitavati isto kao i podaci koji se obrađuju. Primeri prvih ovakvih računara su EDVAC, Mark 1 i EDSAC.

Osnovni elementi fon Nojmanove arhitekture su:

1. procesor - koji čine aritmetičko-logička jedinica, kontrolna jedinica i registri, i
2. glavna memorija

koji su međusobno povezani, dok se ostale komponente računara smatraju pomoćnim. Prikaz fon Nojmanove arhitekture je na slici 5. Pod pomoćne komponente ubra-



Slika 5: Šematski prikaz fon Nojmanove arhitekture

jamo ulazno-izlazne jedinice, spoljašnje memorije itd., koje da bi funkcionisale moraju biti povezane na centralni deo računara (procesor i glavnu memoriju). Osnovna uloga procesora je obrada podataka, dok je osnovna uloga memorije skladištenje podataka koji se obrađuju, kao i programa. Postoji jedinstven način na koji zapisujemo i podatke i programe, a to je uz pomoć nula i jedinica, odnosno, binarnim zapisom. Tokom rada računara podaci i programi se prenose između procesora i glavne memorije.

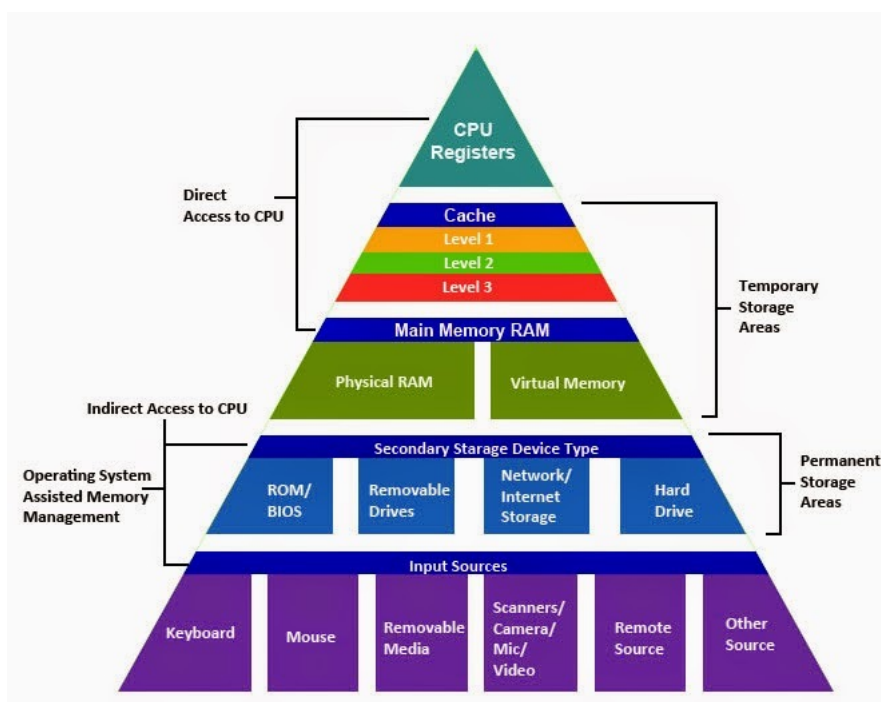
**Procesor** Prva centralna komponenta fon Nojmanove arhitekture. Procesor, koji je odgovoran za rad računara, sastoji se od *kontrolne jedinice* - koja

upravlja radom procesora i *aritmetičko-logičke jedinice* - koja je zadužena za izvođenje aritmetičkih operacija (sabiranje, oduzimanje, množenje, poređenje, itd. ) i logičkih operacija (konjunkcija, negacija, itd. ) nad brojevima. Osim dva navedena dela procesor sadrži i određeni broj registara, obično fiksirane širine (8, 16, 32 ili 64 bita), koji privremeno mogu da čuvaju podatke. Danas procesori neretko poseduju više jezgara (engl. core) koja istovremeno izvršavaju instrukcije čime se obezbeđuje paralelno izvršavanje.

**Memorija** Druga centralna komponenta fon Nojmanove arhitekture je glavna memorija. Memorija predstavlja linearno uređeni niz registara, pri čemu svaki ima svoju adresu. Kao posledica osobine ove memorije da se sadržaju može pristupiti u slučajnom redosledu, čest naziv je i memorija sa slobodnim pristupom (engl. RAM - random access memory). Razlikujemo nekoliko parametara koji odlikuju memoriju, to su:

- kapacitet - GB,
- vreme pristupa - vreme potrebno da se memorija pripremi za čitanje ili upis i
- protok - izražava količinu podataka koji se prenose po jedinici merenja (danas obično mereno u GBps).

Na slici 6 može se videti memorijska hijerarhija.



Slika 6: Šematski prikaz memorijske hijerarhije



### 1.3 Hardver

Bez obzira na činjenicu da osnovu savremenih računarskih sistema i dalje čini fon Nojmanova arhitektura, za rad računara u današnjem smislu reči potreban je i čitav niz hardverskih komponenti koje nam dodatno olakšavaju. Opis komponenti koje čine jedan računar danas ne sastoji se od kućišta, monitora, tastature i miša, već nam je potreban apstraktniji, sveobuhvatniji opis. Upravo, kako bi se jasnije i preciznije opisao računar kaže se da ga čine:

- procesor tj. centralna procesorska jedinica (engl. Central Processing Unit, CPU), koja obrađuje podatke,
- glavna memorija (engl. main memory), u kojoj se istovremeno čuvaju i podaci koji se obrađuju i trenutno pokrenuti programi, i
- različiti periferijski uređaji ili ulazno-izlazne jedinice (engl. peripherals, input-output devices, IO devices), u koje se ubrajaju miševi, tastature, ekrani, štampači, diskovi, a koji služe za interakciju između korisnika i sistema i trajno skladištenje podataka i programa.

Da bi se izvršilo povezivanje svih navedenih komponenti koristimo magistralu. Za funkcionisanje modernih računara neophodni su i hardver i softver. Hardver (tehnički sistem računara) čine opipljive, fizičke komponente računara: procesor, memorija, matična ploča, itd.

### 1.4 Softver

Softver računara čine programi i prateći podaci koji određuju izračunavanja koja vrši računar. Prvi računari su se odlikovali jezicima specifičnim za konkretni računar - mašinski zavisnim jezicima. Već od 1950-ih, sa pojavom prvih jezika višeg nivoa, programiranje postaje dosta lakše. Danas, programi se najčešće pišu u višim programskim jezicima, a potom se prevode na mašinski jezik, onaj koji je razumljiv računaru. Programom opisujemo računaru koje operacije treba da izvrši sa ciljem ispunjavanja nekog zadatka. U nastavku biće navedeno nekoliko primera koji ilustruju izvršavanje programa napisanih na višim programskim jezicima.

**Primer 1.1** *Želimo da izračunamo vrednost izraza  $2*x+3$  za neko  $x$ . Podatke u računarstvu, kao i u matematici, možemo predstaviti pomoću promenljivih. Međutim, za razliku od matematike, promenljive u računarstvu mogu menjati svoju vrednost. Svakoj promenljivoj je u memoriji računara pridruženo jedno fiksirano mesto i ona može tokom izvršavanja programa da menja vrednost. Recimo da je  $x$  ulazni parametar našeg programa, a  $y$  izlazna vrednost, tada izračunavanje opisujemo sa*

$y := 2*x + 3$

*gde  $*$  označava množenje,  $+$  sabiranje, a  $:=$  naredbu dodele, odnosno promenljivoj sa leve strane izraza dodeljujemo vrednost izraza sa desne strane.*

**Primer 1.2** *Kao naredni primer uzmimo poređenje dva broja, odnosno, kao izlaz treba da dobijemo veći od dva broja. Ovakav izraz možemo zapisati kao:*

```

ako je x >= y onda
    m := x
inače
    m := y

```

**Primer 1.3** *Kao još jedan primer uzmimo stepenovanje:*

```

s := 1, i := 0
dok je i < n radi sledeće:
    s := s·x,
    i := i + 1

```

Savremeni softver klasifikujemo u 2 kategorije:

- Sistemski i
- Aplikativni.

**Aplikativni softver** je onaj koji krajnji korisnici računara direktno koriste u svojim svakodnevnim aktivnostima. Tu spadaju, na primer, pregledači Veba, e-mail klijenti, kancelarijski softver (programi za kucanje teksta, izradu prezentacija,...), video igre, softver za prikaz slika, itd.

**Sistemski softver** ima ulogu da kontroliše hardver i pruža usluge aplikativnom softveru. Najznačajniji skup sistemskog softvera je operativni sistem (OS), ali u sistemski softver ubrajamo i različite uslužne programe: editore teksta, alate za programiranje (prevodioci, dibageri, profajleri, integrisana okruženja) i slično. Uloga operativnog sistema je da programeru pruži skup funkcija koje programer može da koristi kako bi ispunio određeni cilj, sakrivajući konkretne hardverske detalje - ovaj skup funkcija naziva se programski interfejs za pisanje aplikacija (engl. API - Application Programming Interface).

## 1.5 Oblasti savremenog računarstva

Savremeno računarstvo sastoji se iz više podoblasti između kojih nema jasnih granica. Prema klasifikaciji američke asocijacije ACM - Association for Computing Machinery, razlikujemo naredne podoblasti [?]:

- Algoritmika (procesi izračunavanja i njihova složenost)
- Strukture podataka (reprezentovanje i obrada podataka)
- Programski jezici (dizajn i analiza svojstava formalnih jezika za opisivanje algoritama)
- Programiranje (proces zapisivanja algoritama u nekom programskom jeziku)
- Softversko inženjerstvo (proces dizajniranja, razvoja i testiranja programa)
- Prevođenje programskih jezika (efikasno prevođenje viših programskih jezika, obično na mašinski jezik)
- Operativni sistemi (sistemi za upravljanje računarom i programima)
- Mrežno računarstvo (algoritmi i protokoli za komunikaciju između računara)

- Primene (dizajn i razvoj softvera za svakodnevnu upotrebu)
- Istraživanje podataka (pronalaženje relevantnih informacija u velikim skupovima podataka)
- Veštačka inteligencija (rešavanje problema u kojima se javlja kombinatorna eksplozija)
- Robotika (algoritmi za kontrolu ponašanja robota)
- Računarska grafika (analiza i sinteza slika i animacija)
- Kriptografija (algoritmi za zaštitu privatnosti podataka)
- Teorijsko računarstvo (teorijske osnove izračunavanja, računarska matematika, verifikacija softvera, itd).

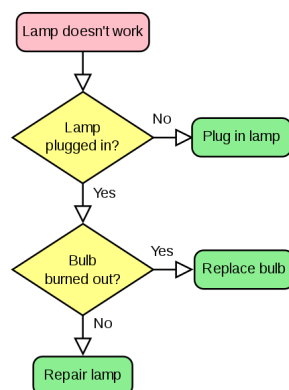
## 1.6 Osnovni pojmovi i konstrukti

Programiranje predstavlja proces zapisivanja algoritama u nekom programskom jeziku. Algoritam predstavlja precizan opis postupka za rešavanje nekog problema u konačnom broju koraka. Algoritmi se odlikuju svojom složenošću. Ta složenost može biti vremenska ili memorijska. Vremenska složenost se odnosi na vreme potrebno za izvršavanje nekog algoritma. Memorijska složenost označava koliko memorijskih resursa je potrebno za izvršavanje algoritma. Cilj analize algoritama je predviđanje njegovog ponašanja i brzine izvršavanja bez realizacije na nekom konkretnom računaru. Ta procena treba da se odnosi na svaki računar. Nemoguće bi bilo na svakom računaru ispitati izvršavanje nekog algoritma. Zbog toga je analiza algoritama približna tehnika. [?] Postoji uniformna tehnika za grafički prikaz algoritama, međutim, mi nećemo ulaziti u detalje, već će biti dato nekoliko ilustrativnih primera kako bi se dobila glavna ideja.

**Primer 1.4** *Kao najjednostavniji primer algoritma uzmimo primer sa lampom. U početnom koraku lampa je ugašena, a mi želimo da je upalimo ili, ako ne radi, da je odnesemo na popravku. Najpre, proveravamo da li je lampa uključena u struju. Romb je znak kojim označavamo uslov. Ako lampa nije uključena u struju treba je uključiti. U suprotnom, ako je lampa uključena u struju, ali ne svetli, proveravamo da li je sijalica pregorela. Ako jeste, menjamo sijalicu. Ako nije, lampa je pokvarena i moramo je popraviti. Na slici 7 vidimo kako bismo grafički prikazali opisani postupak.*

**Primer 1.5** *Kao primer, više prikladan računarskoj terminologiji, uzmimo sabiranje 2 broja,  $M$  i  $N$ . Na slici 8 vidimo kako bismo grafički prikazali postupak sabiranja brojeva.*

U sekciji 1.2 koja govori o fon Nojmanovoj arhitekturi, videli smo da se podaci (i programi) smeštaju u memoriju računara, najčešće u vidu niza bitova. Međutim, kako se programer ne bi zamarao detaljima i kako bi mogao da radi na dosta apstraktnijem nivou, programski jezici obezbeđuju koncept promenljivih. Promenljive daju mogućnost programeru da podacima dodeli imena i da im na osnovu tih imena i pristupa. Svaka promenljiva u programu ima dodeljen određeni niz bajtova.



Slika 7: Primer najjednostavnijeg algoritma.



Slika 8: Prikaz algoritma za sabiranje dva broja.

Promenljive se odlikuju svojim tipovima i životnim vekom. Životni vek je koncept koji nam govori u kom delu faze izvršavanja programa je promenljivoj dodeljen memorijski prostor, odnosno, kada je možemo koristiti. Životni vek promenljive omogućava da na različitim mestima u programu koristimo različite promenljive istog imena i pravilo doseg identifikatora (engl. scope) određuje deo programa u kome se uvedeno ime može koristiti. Druga odlika promenljivih su tipovi. Organizovanje podataka u tipove pomaže programeru da ne mora da razmišlja o podacima na nivou njihove binarne reprezentacije, već daleko apstraktnije. Neki od najčešćih tipova su:

- celi brojevi (... -3, -2, -1, 0, 1, 2, 3,...),
- brojevi u pokretnom zarezu ( 1.0, 3.14, 9.81,...),
- karakteri (a, b, P, „ !, ...),
- niske (ždravo", švima",...)

Osim ovih postoje i složeniji tipovi koji mogu objediniti više istih ili različitih tipova, pa tako imamo nizove, strukture, liste, i dr. Svaki tip podataka se karakteriše vrstom podataka koje opisuje, skupom operacija koje se nad njime vrše i načinom reprezentacije i detaljima implementacije tog tipa.

Osnovni gradivni elementi imperativnih programskih jezika su naredbe. Naredba dodele je osnovna naredba i njom se vrednost neke promenljive postavlja na vrednost nekog izraza definisanog nad konstantama i promenljivim. Šta to, u praksi, znači? To znači da kada kažemo  $x = 3 * y$  zapravo promenljivoj  $x$  dodeljujemo vrednost  $3 * y$ . Naredbe se u programu nižu jedna za drugom,

osim u slučaju korišćenja naredbi za kontrolu toka izvršavanja programa. Ove naredbe u zavisnosti od tekućih vrednosti promenljivih neke naredbe mogu da ne izvršavaju, izvršavaju ih više puta (petlje) i slično. Najčešće korišćene kontrolne strukture su granajuće naredbe (if-then-else), petlje (for, while, do-while, repeat-until) i naredbe skoka (goto). Naredba if-then-else ima sledeći opšti oblik:

```
if izraz1
    naredba1
else
    naredba2
```

Treba obratiti pažnju da je else grana neobavezna, odnosno, može, ali ne mora, da postoji.

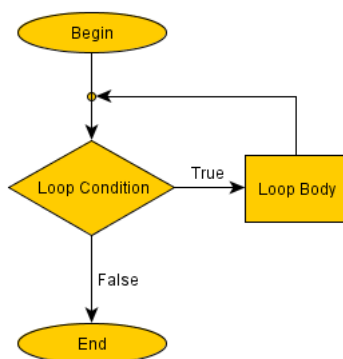
**Primer 1.6** *Naredni primer štampa veći od dva broja.*

```
if a > b
    print(a)
else
    print(b)
```

**Primer 1.7** *Naredni primer u promenljivu a smešta tekst "Hello world!" ako je vrednost promenljive b veća od 0.*

```
if b > 0
    a = "Hello world!"
```

Jedan od najbitnijih koncepata programiranja je pojam *petlje*. Petlje predstavljaju konstrukcije koje nam omogućuju da izvršavamo jednu ili više akcija više puta. Sastoje se iz uslova i tela petlje. Uslov nam definiše u kom slučaju ćemo napustiti petlju (na primer, ako broj iteracija<sup>7</sup> petlje predje 100, ili ako je neka promenljiva n veća od 1000, itd.). Telo petlje sadrži akcije koje želimo da ponavljamo. Na slici 9 može se videti opšti oblik petlje.

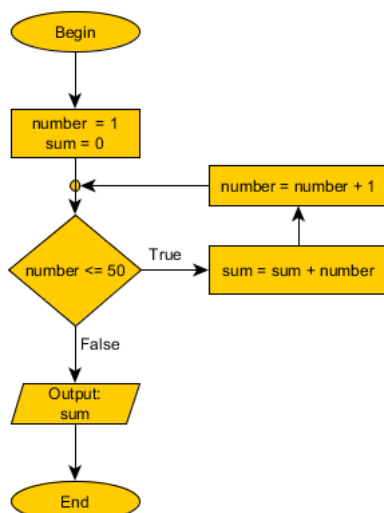


Slika 9: Petlja.

---

<sup>7</sup>Iteracija petlje predstavlja jedno izvršavanje koda koji se nalazi u telu petlje.

**Primer 1.8** Za naredni primer, uzećemo sabiranje prvih 50 brojeva. Naime, naporno bi bilo da pišemo  $1+2+3+\dots+50$ , a da ne pomišljamo na brojeve poput 1000, 100000 ili 1000000. To bi bilo gotovo neizvodivo. Zbog toga, želimo da na apstraktniji način opišemo proceduru koja će umesto nas izvršiti sabiranje prvih 50 brojeva. Da bismo to uspeali moramo iskoristiti petlju. Na slici 10 vidimo kako bismo grafički prikazali postupak.



Slika 10: Prikaz algoritma za sabiranje prvih 50 brojeva.

Petlje su neophodne za uspešno programiranje. Postoji nekoliko različitih vrsta petlji, čija upotreba zavisi od onoga što njom želimo da postignemo, pa tako imamo:

- "for" petlju i
- "while" petlju

koje predstavljaju osnovne konstrukte u skoro svim programskim jezicima. Osim for i while petlje postoje i druge, ali o njima neće biti reči.

For petlja je najčešće oblika:

```
for(izraz1, izraz2, izraz3)
    naredba
```

*Izraz1* i *izraz3* obično predstavljaju naredbe dodele ili inkrementiranje, gde se *izraz1* obično naziva inicijalizacija, a *izraz3* je korak. Izraz u sredini, *izraz2*, predstavlja relacijski izraz i služi kao uslov izlaska iz petlje. *Naredba* predstavlja liniju ili blok koda koji želimo da se izvršava. Kao posledica navedenog, *for* petlju možemo posmatrati kao:

```
for(inicijalizacija, uslov izlaska, korak)
    kod koji želimo da se izvrši
```

**Primer 1.9** *Naredni primer predstavlja jedan od uobičajenih oblika u kojima se for petlja pojavljuje:*

```
for(i = 0; i < n; i++)  
    print(i)
```

*Ovaj primer za svaki korak petlje ispisuje broj koraka.*

## 1.7 Klasifikacija programskih jezika

Brojnost programskih jezika raste iz godine u godinu. Stalno se pojavljuju novi i brži jezici, unapređuju se stari i nemoguće je ispratiti sve promene. Da bismo odmah razumeli okvirno kako neki programski jezik funkcioniše moramo znati kojoj paradigmi pripada. Kada kažemo da neki jezik pripada nekoj paradigmi mi zapravo govorimo nešto o karakteristikama tog jezika koje važe za sve jezike koji pripadaju istoj grupi. Programske paradigme su formirane prema načinu programiranja. Neki od najkorišćenijih programskih jezika današnjice spadaju u grupu imperativnih programskih jezika, npr. jezik C. Glavna karakteristika imperativnih programskih jezika je da stanje programa karakterišu promenljive kojima se predstavljaju podaci i naredbe kojima se vrše određene transformacije nad promenljivim (sabiranje, oduzimanje, poređenje, itd.). Osim imperativne, značajne programske paradigme su i objektno-orijentisana (C++, Java (Java NIJE JavaScript!), C# itd.), funkcionalna (Lisp, Haskell, ML, itd.), logička (u nju spada, na primer, Prolog). Sa razvojem savremenih programskih jezika došlo je do brisanja jasnih granica između ovih jezika, pa tako dolazi do mešanja karakteristika različitih paradigmi.

Za većinu programskih jezika danas reći ćemo da su proceduralni. Kada kažemo da je neki jezik proceduralan zapravo želimo da iskažemo činjenicu da je zadatak programera da opiše način (proceduru) kojim će se doći do rešenja problema. Kao idejno potpuno kontrastni, postoje deklarativni programski jezici (poput Prologa) koji od programera zahteva precizan opis problema, a mehanizam programskog jezika se onda bavi pronalaskom rešenja.

Prema tipu konverzije tipova imamo statički i dinamički tipizirane jezike. Kod statički tipiziranih jezika (poput C-a) zahteva se da programer definiše tip svake promenljive i da ga potom više ne menja tokom izvršavanja programa. Kod dinamički tipiziranih jezika ista promenljiva može sadržati podatke različitog tipa tokom različitih faza izvršavanja. Nekada je moguće čak i vršenje operacija nad promenljivima različitog tipa, pri čemu dolazi do implicitne konverzije. Na primer, jezik JavaScript ne zahteva definisanje tipa promenljivih i dopušta kôd poput  $a = 1; b = "2"; a = a + b;$

## 1.8 Primeri za vežbu

**Primer 1.10** *Napisati primer algoritma za kuvanje kafe.*

**Primer 1.11** *Napisati primer algoritma za provodjenje jednog dana.*

**Primer 1.12** *Napisati primer algoritma za savladavanje nekog kursa.*

**Primer 1.13** *Napisati primer algoritma za računanje zbira prvih 10 parnih brojeva.*

**Primer 1.14** *Koju vrednost ima promenljiva  $x$  nakon izvršavanja narednog koda:*

```
int x = 0;
if (x > 3);
    x++;
```

**Primer 1.15** *Napisati pseudo kod za računanje zbira 3 broja.*

**Primer 1.16** *Napisati pseudo kod u kome se promenljivoj  $a$  dodeljuje vrednost 3, promenljivoj  $b$  dodeljuje vrednost 6 i onda se njihov zbir smešta u promenljivu  $c$ . Nakon toga, proveriti da li ostatak pri deljenju promenljive (računa se uz pomoć  $\%$ )  $c$  sa brojem 2 daje 0, odnosno, da li je  $c$  paran.*

**Primer 1.17** *Napisati pseudo kod<sup>8</sup> algoritma za računanje zbira prvih 10 parnih brojeva.*

**Primer 1.18**

## 1.9 Domaći zadatak

Domaći zadatak:

- pročitati nešto dodatno o istorijatu računarstva,
- opisati ukratko svaki od elemenata memorije,
- odraditi sve primere i pitalice,
- samostalno pronaći još 10 novih i uraditi ih.

---

<sup>8</sup>Pseudo kod predstavlja kod koji nije dat u formalnim terminima, već predstavlja ideju kako bi kod trebao da izgleda.



## 2 Uvod u web

Danas, ne možemo zamisliti korišćenje računara bez veza ka drugim računarima. Izgradnja računarskih mreža, a posebno sa nastankom i razvojem Interneta i njegovih servisa poput Veba, dovele su do porasta broja korisnika računara i promene uloga računara u odnosu na ranije. Pojava savremenih računarskih mreža smatra se revolucionarnom poput pojave parne mašine u 18. veku. Svake godine uvećava se broj umreženih računara, a sa tim brojem raste i broj usluga koje nam mrežno okruženje nudi. Neke od osnovnih primera upotreba računarskih mreža obuhvataju:

- poslovna: elektronska pošta, razmena datoteka, deljeni štampači, ...
- kućna: filmovi, muzika, igrice, vesti, audio i video komunikacija, razmena poruka, elektronska kupovina,...
- mobilne: pozivi, SMS, igrice, mape, pristup informacijama

### 2.1 Uloga računarskih mreža

U osnovne uloge računarskih mreža ubrajamo:

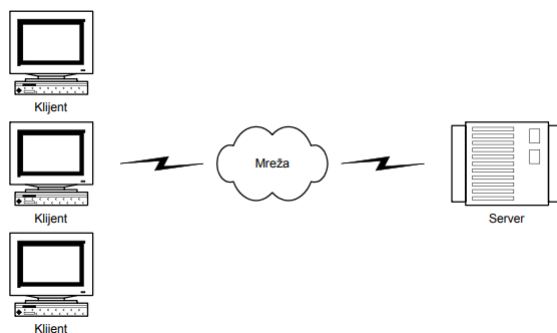
1. komunikaciju - uz pomoć računara ljudi razmenjuju poruke, video pozive, mejlove, ćaskanja (eng. chat), video konferencije, itd.
2. deljenje informacija i podataka - ako postoji mrežno okruženje u kom su računari povezani, tada je moguće pristupiti informacijama na drugim računarima u okviru mreže. Podatke prenosimo na više načina, kao što su preuzimanje datoteka, prenos informacija u okviru lokalnih mreža (obično u okviru jedne kompanije), kao i u okviru globalne svetske mreže. Internet i veb se smatraju glavnim izvorima informacija.
3. deljenje softvera - korisnici povezani u mrežu mogu koristiti mnoge usluge koje im pruža softver koji radi na računarima u okviru mreže. Neke od usluga su kupovina i rezervacija karata preko interneta, ili izvršavanje softvera koji je distribuiran i paralelizovan na više povezanih računara, čime se može ubrzati izvršavanje.
4. deljenje hardverskih resursa - obezbeđuje zajedničko korišćenje hardvera, poput štampača, skenera i ostalih. Često se ovakav pristup koristi u kompanijama.

Računarski resursi u mreži mogu biti raspoređeni na različite načine, tako da obezbeđuju različite načine izvršavanja poslova. Neki od najčešćih su:

- Centralizovana obrada - svi poslovi se izvršavaju na jednom centralnom računaru, dok se ostali uređaji u mreži koriste samo kao terminali za unos podataka i prikaz rezultata. Ovakvim načinom rada odlikovale su se rane računarske mreže.
- Klijent-server okruženje - jedan računar ima ulogu servera na kome se nalaze podaci i aplikativni softver, koji se stavljaju na raspolaganje klijentima. Serveri su obično moćniji računari od klijenata (mada ne mora uvek biti tako) i na njima se obavljaju zadaci koji zahtevaju više resursa.

U današnjem kontekstu, stroga podela na klijentski i serverski računar više nije tako aktuelna. Najčešće govorimo o tome da je jedan računar istovremeno i klijent i server u zavisnosti od zadataka koji su mu zadati. Na primer, isti računar može istovremeno pokretati i Veb server i klijent za elektronsku poštu, čime mu je data i uloga servera i uloga klijenta. na njihov zahtev.

- Mreža ravnopravnih računara (eng. peer-to-peer - *P2P*) - računari direktno komuniciraju deleći podatke i opremu. Sve se češće ovakve mreže koriste za masovnu razmenu velikih količina podataka (npr. torrenti - Bit-torrent).



Slika 11: Prikaz klijent-server arhitekture.

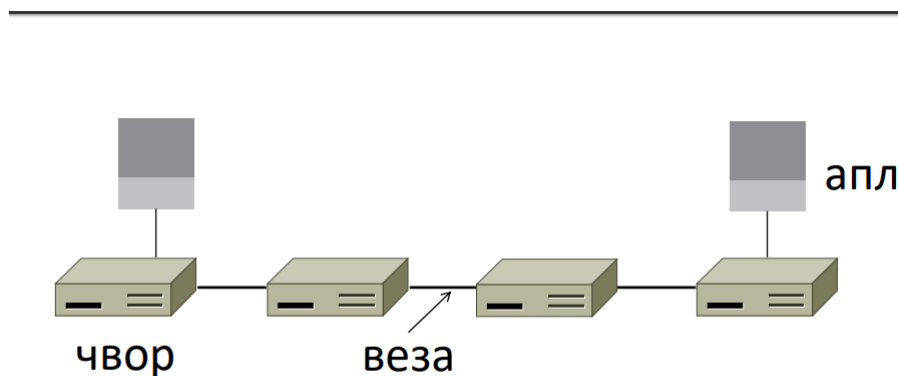
## 2.2 Komponente računarskih mreža

Pre nego što uđemo u detaljniji opis elemenata koji čine jednu računarsku mrežu, trebalo bi da damo formalnu definiciju šta je računarska mreža. Naime, računarska mreža je sistem koji se sastoji iz skupa hardverskih uređaja koji su međusobno povezani komunikacijskom opremom i koji je snabdeven odgovarajućim kontrolnim softverom kojim se ostvaruje kontrola funkcionisanja sistema tako da je moguć prenos podataka između povezanih uređaja. Neke od osnovnih komponenti računarskih mreža su, dakle:

- mrežni hardver,
- komunikacioni kanali i
- mrežni softver.

### 2.2.1 Mrežni hardver

Tradicionalno, podrazumeva se povezivanje računara u okviru mreže, ili uz dodatak nekih pomoćnih uređaja poput štampača, skenera itd., kako bi se mogli deljeno koristiti. Međutim, u poslednje vreme, granica između klasičnih računara i digitalnih uređaja specijalizovane namene se briše i sve češće se u okviru mreže mogu povezati i PDA uređaji, mobilni telefoni, foto aparati, kamere



Slika 12: Prikaz mreže.

i ostali. Aktivno se radi i na razvoju automobila, frižidera i ostalih mnogobrojnih uređaja kako bi se uključili u mrežu i kako bi se time omogućilo upravljanje istima na daljinu.

Da bismo neki uređaj mogli da ubacimo u mrežu, neophodno je da sadrži određene hardverske komponente koje bi mu to omogućile. Deo hardvera koji je namenjen za umrežavanje i spada u komunikacionu opremu je mrežna kartica ili mrežni adapter (eng. NIC - network interface card) koja omogućava fizički pristup mreži. Svaka mrežna kartica ima svoju jedinstvenu fizičku (MAC) adresu, kojom se uređaj jedinstveno identifikuje prilikom komunikacije. Neke mrežne kartice obezbeđuju pristup žičanim, a neke druge bežičnim komunikacionim kanalima. Osim mrežnih kartica za umrežavanje se koriste i modemi (telefonski, kablovski), kao i još neki uređaji o kojima neće biti reči.

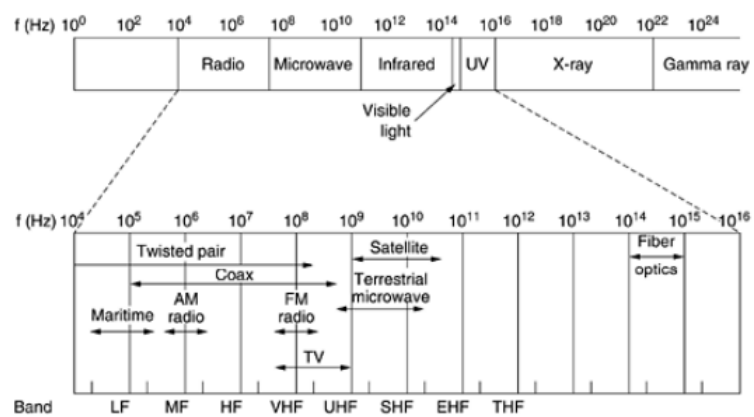
### 2.2.2 Komunikacioni kanali

Da bi mreža funkcionisala i da bi kroz nju bilo moguće preneti podatke, uređaji koji se nalaze u mreži moraju biti povezani međusobno uz pomoć žičanih ili bežičnih prenosnih sistema, koji predstavljaju komunikacione kanale. Osnovna mera kvaliteta komunikacionog kanala je brzina prenosa koja se meri u bitovima po sekundi (bit/s). Ova mera označava broj bitova koji se mogu preneti kroz komunikacioni kanal u jednoj sekundi. Ako bismo posmatrali aktuelne tehnologije prenosa podataka, najčešće se koriste megabiti ( milion bita) u sekundi - Mbps, ili gigabiti (milijarda bita) u sekundi - Gbps. Brzina prenosa predstavlja fizičku karakteristiku komunikacionog kanala i zavisi od frekvencijskog opsega (eng. bandwidth) koji se može propustiti kroz kanala bez gubitka signala. Na slici 14 prikazan je raspon frekvencija za razne prenosne tehnologije.

Komunikacione kanale možemo podeliti u dve grupe prema tipu prenosa



Slika 13: Prikaz mrežne kartice.

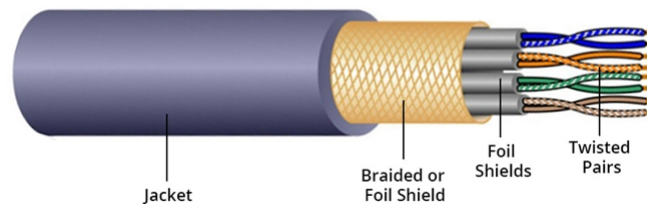


Slika 14: Prikaz frekvencijskih opsega za razne prenosne tehnologije.

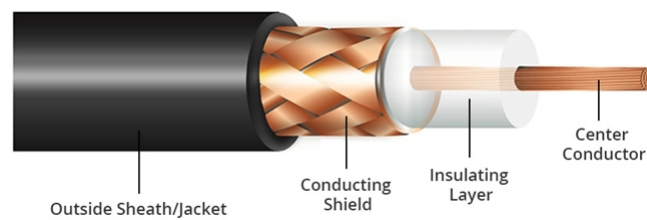
informacija:

1. Žičane
2. Bežične

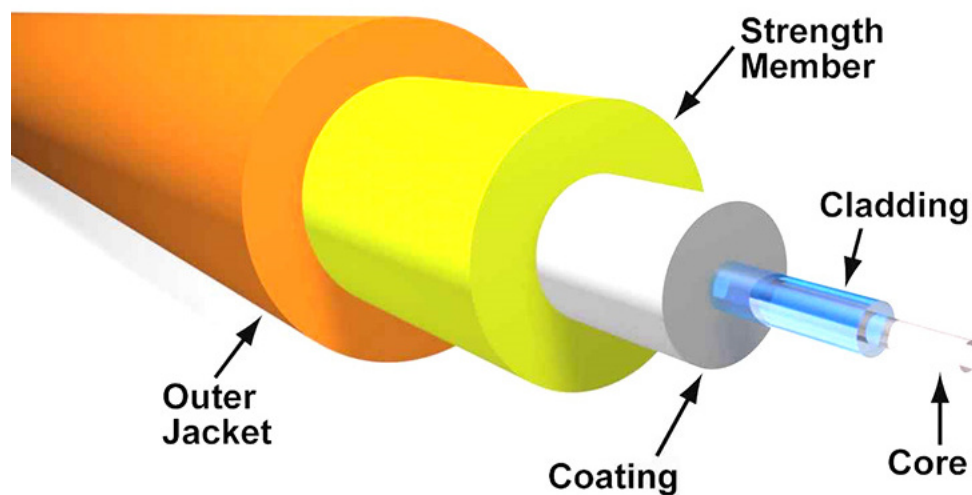
**Žičane komunikacije** Parice (eng. twisted-pair wire) - Najkorišćeniji način komunikacije. Uređaji se povezuju korišćenjem uvijenih uparenih izolovanih bakarnih žica. Žice se uparuju i uvijaju kako bi se smanjile smetnje u komunikaciji. Brzina prenosa kroz ovakav medijum obično varira od  $2Mbps$  do  $100Mbps$ .



Slika 15: Prikaz parice.



Slika 16: Prikaz koaksijalnog kabla.



Slika 17: Prikaz optičkog kabla.

### **2.2.3 Mrežni softver**

### 3 HTML i CSS

**4 Zaključak**

**A Dodatak**