Uvod u programiranje

Skipta za kurs
Uvod u programiranje kroz JavaScript

Una Stanković
una.stankovic@code.edu.rs

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

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, tehnici 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 jedne 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[1] je 1642. godine konstruisao mehaničke sprave koje su služile za sabiranje i oduzimanje celih brojeva, zvane Paskaline. Trideset godina nakon njega, Godfrid Lajbnic[2] konstruisao je mašinu, zasnovanu na dekadnom sistemu, koja je mogla da vrši sve četiri osnovne operacije. Lajbnic je bio prvi koji je predlagao koriscenje binarnog sistema.

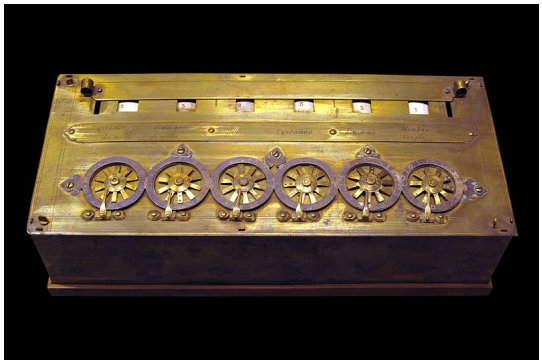
Mehaničke mašine Žozef Mari Žakard[3] je 1801. godine napravio prvu programabilnu mašinu — mehanički tkački razboj. On je pomoću bušenih kartica kreirao kompleksne šare na tkanini. Svaka rupa na kartici određivala je

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

[2] Gottfried Wilhelm Leibniz (1646–1716), nemački filozof i matematičar

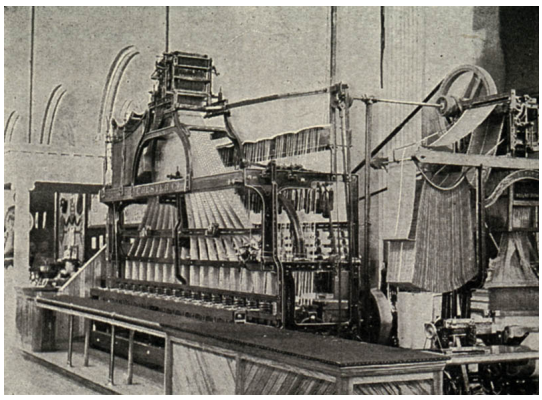
[3] Joseph Marie Jacquard (1752–1834), francuski trgovac.

Slika 1: Abakus



Slika 2: Paskalina

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

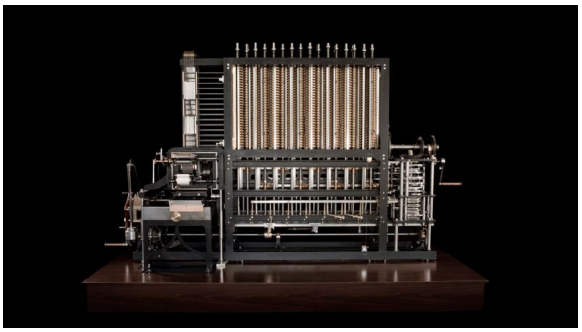


Slika 3: Žakardov razboj

prvoj polovini 19. veka, Čarls Bebidž[4] 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.

[4]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 računuar 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[5] 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.[6]

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

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

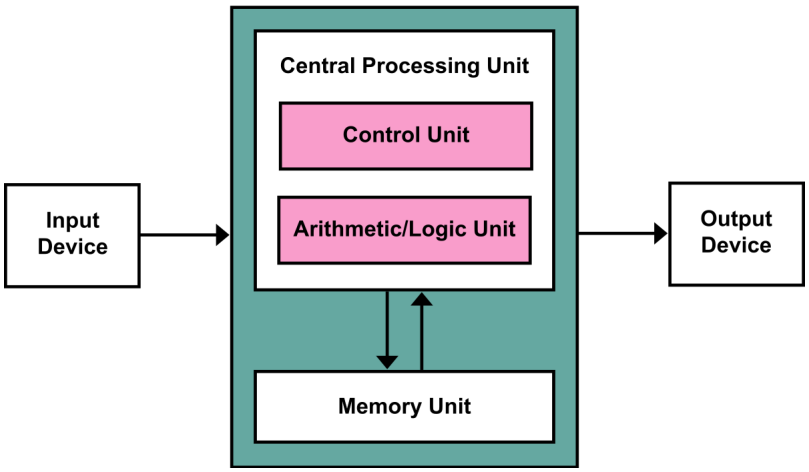
[6]Herman Hollerith (1860–1929), američki pronalazač.

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 jediice, 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.

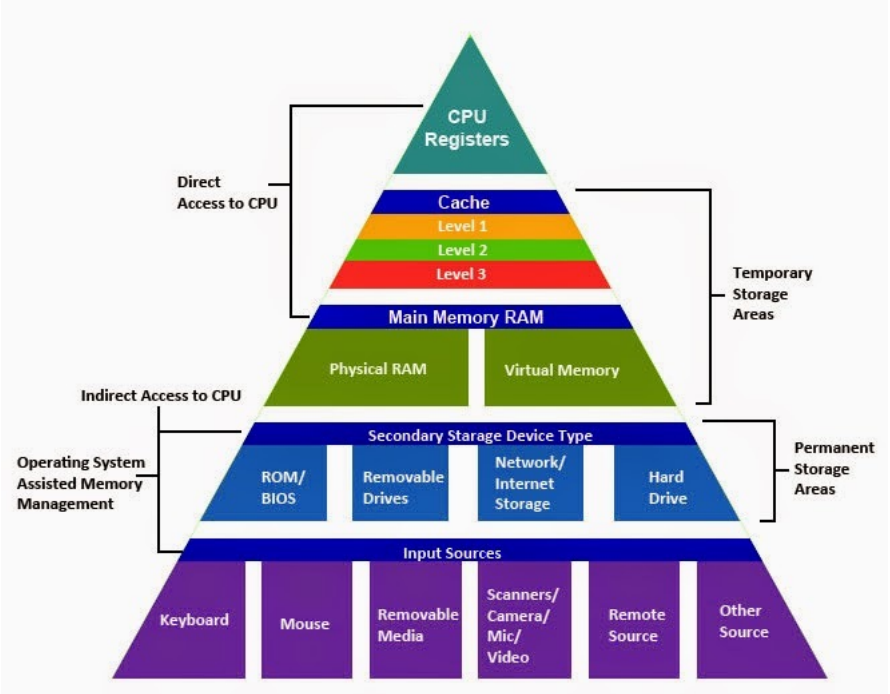
Procesor Prva centralna komponenta fon Nojmanove arhitekture. Procesor, koji je odgovoran za rad računara, sastoji se od *kontrolne jedinice* - koja

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 pristupati 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).

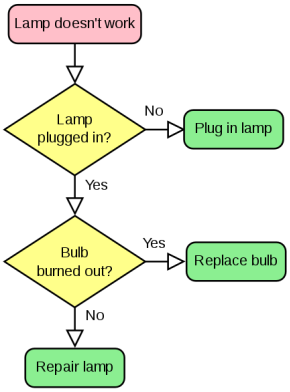
1.6 Osnovni pojmovi i konstrukti

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.

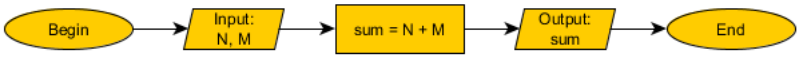
Primer 1.4 *Kao najjednostavniji primer algoritma uzmimo primer sa lampom. U početnom koraku lampa je ugašena, a mi želimo da je upalimo ili, ako ne radi, da je odnesemo na popravku. Najpre, proveravamo da li je lampa uključena u struju. Romb je znak kojim označavamo uslov. Ako lampa nije uključena u 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.

Promenenljive 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 dosega 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.

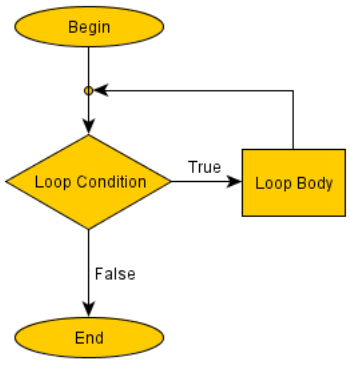
Primer 1.6 *Naredni primer štampa veći od dva broja.*

```
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!"
```

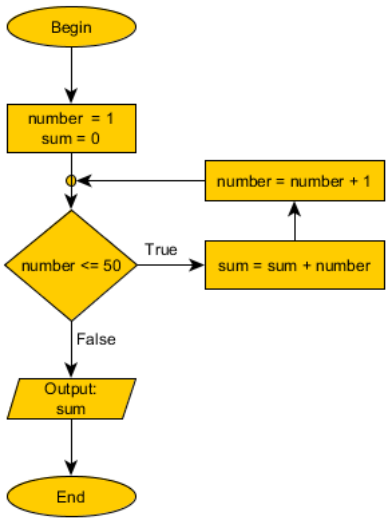
Jedan od najbitnijih koncepata programiranja je pojam *petlje*. Petlje predstavljaju konstrukcije koje nam omogućuju da izvršavamo jednu ili više akcija više puta. Sastoje se iz uslova i tela petlje. Uslov nam definiše u kom slučaju ćemo napustiti petlju (na primer, ako broj iteracija[7] petlje predje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



Slika 9: Petlja.

[7]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* $1+2+3+...+50$*, 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 uspeli 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 nece 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 promenjlivih 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*[8] *algoritma za računanje zbira prvih 10 parnih brojeva.*

Primer 1.18

1.9 Domaći zadatak

Domaći zadatak:

- pročitati nešto dodatno o istorijatu računarstva (01_istorijat),
- opisati ukratko svaki od elemenata memorije (01_piramida),
- odraditi sve primere i pitalice ,
- domaci sa osi/tcp slojevima: 01_osi, 01_tcp
- domaci koji je size teksta o browserima: 01_browser
- samostalno pronaći još 10 novih i uraditi ih(01_0,...,01_9).

[8] 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:

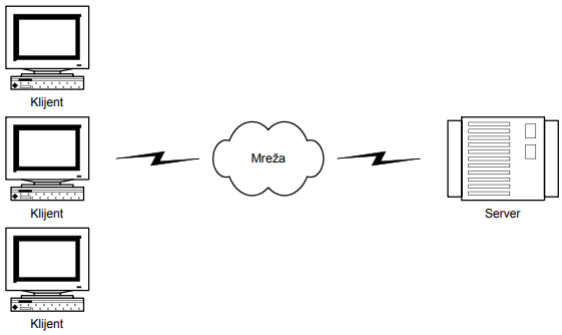
1. komunikaciju - uz pomoć računara ljudi razmenjuju poruke, video pozive, mejlove, ćaskanja (eng. chat), video konferencije, itd.
2. deljenje informacija i podataka - ako postoji mrežno okruženje u kom su računari povezani, tada je moguće pristupiti informacijama na drugim računarima u okviru mreže. 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. Ovakavim 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. torenti - Bittorrent).



Slika 11: Prikaz klijent-server arhitekture.

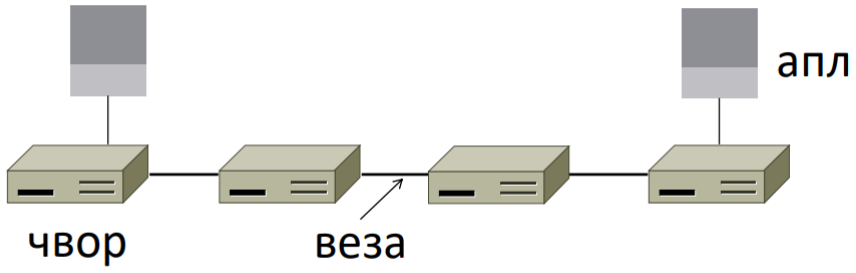
2.2 Komponente računarskih mreža

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 oviru mreže mogu povezati i PDA uređaji, mobilni telefoni, foto aparati, kamere



Slika 12: Prikaz mreže.

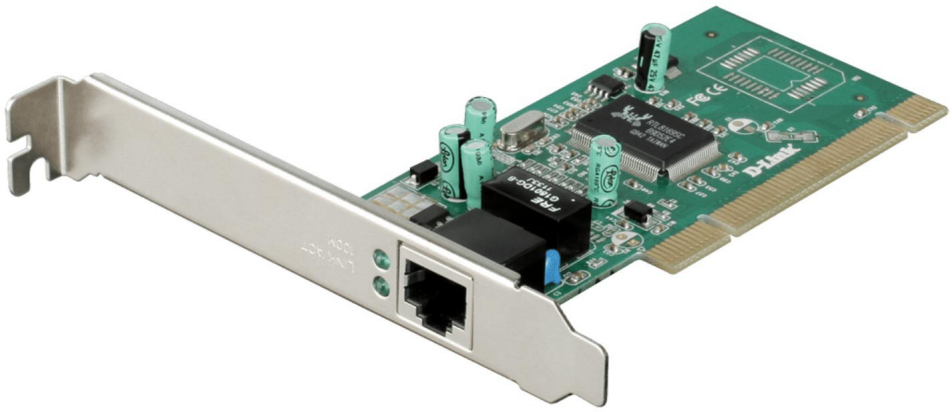
i ostali. Aktivno se radi i na razvoju automobila, frižidera i ostalih mnogobrojnih uređaja kako bi se uključili u mrežu i kako bi se time omogućilo upravljanje istima na daljinu.

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.

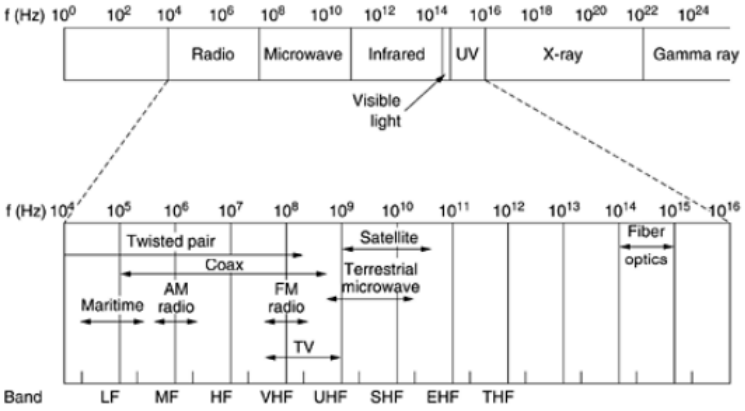
2.2.2 Komunikacioni kanali

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 prikayan je raspon frekvencija za razne prenosne tehnologije.

Komunikacione kanale možemo podeliti u dve grupe prema tipu prenosa



Slika 13: Prikaz mrežne kartice.



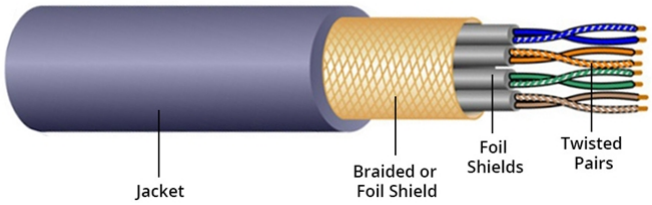
Slika 14: Prikaz frekvencijskih opsega za razne prenosne tehnologije.

informacija:

1. Žičane
2. Bežične

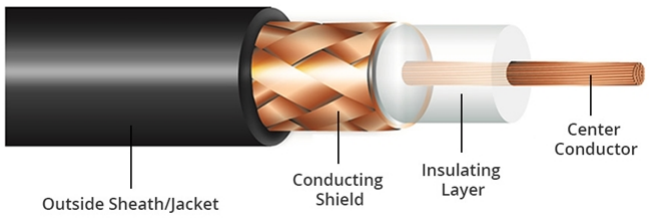
Žičane komunikacije

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.

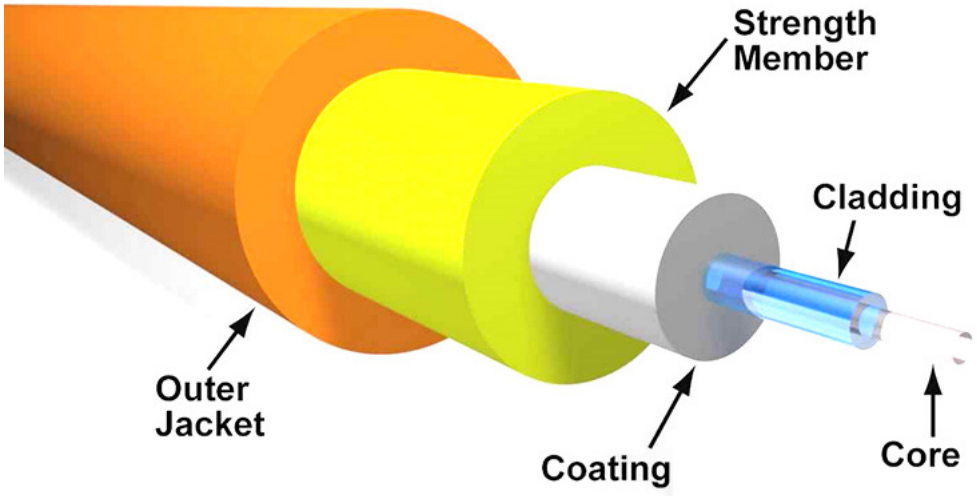
Koaksijalni kablovi svoju upotrebu najčešće nalaze u televizijskim kablovskim sistemima, a koriste se i u lokalnim mrežama u kompanijama. Kablovi se sastoje od centralne bakarne ili aluminijumske žice obmotane savitljivim slojem izolacije, oko kog je obmotan provodni sloj tankih žica, a potom je sve to izolovano. Ovaj tip kablova omogućava brzinu prenosa do $200Mbps$ (nekad čak i do $500Mbps$), uz manju osetljivost na elektromagnetne smetnje.



Slika 16: Prikaz koaksijalnog kabla.

Optički kablovi - prave se od velikog broja (reda veličine nekoliko stotina ili hiljada) veoma tankih staklenih vlakana. Podaci se prenose svetlosnim talasima uz pomoć malog laserskog uređaja. Na ovaj tip kablova elektromagnetne smetnje nemaju uticaja. Najveći nedostatak ovakvih kablova je cena, izuzetno su skupi i komplikovani za komunikaciju, pa se uglavnom koriste za osovinski deo mreže, na koji se potom drugim vrstama kablova povezuju pojedinačni uređaji. Brzina ovih uređaja predstavlja njihovu najveću prednost. Naime, brzina ovih uređaja može ići i do nekoliko triliona bita u sekundi. Najčešće se koriste za mreže sa brzinama do $10Gbps$.

Bežične komunikacije Bežična komunikacija, kao što i samo ime sugeriše, ne koristi kablove za prenos podataka. Ovakav vid komunikacije poseban



Slika 17: Prikaz optičkog kabla.

značaj nalazi kod prenosivih računara, mobilnih telefona ili dosta udaljenih lokacija do kojih bi bilo jako skupo, ako ne i nemoguće, sprovesti kablovsku mrežu. Umesto kablova ove mreže koriste radio talase, mikro talase i infracrvene zrake. Podaci se prenose moduliranjem amplitude, frekvencije ili faze talasa. Neke od danas najkorišćenijih tehnologija su:

- Bluetooth - koristi se za veoma male razdaljine (do 10 ili do 100 metara. Brzina prenosa je do $3Mbps$. Bluetooth tehnologija koristi radio talase i može da prođe i kroz čvrste prepreke. Koristi se najčešće za komunikaciju računara sa periferjskim uređajima, kao i u mobilnoj telefoniji.

- Bežični LAN - Wireless LAN (WLAN, WiFi) je tehnologija koja korsti radio talase za bežičnu komunikaciju više uređaja na ograničenim rastojanjima (nekoliko desetina ili stotina metara). Brzina prenosa ide od $10Mbps$ do $50Mbps$ (u skorije vreme može ići i do $600Mbps$). Najrašireniji standard za ovaj vid komunikacije je IEEE 802.11, o kome će kasnije biti više reči.

- Ćelijski sistemi - Način prenosa je sličan onom koji se koristi u mobilnoj telefoniji. Za komunikaciju se koriste radio talasi i sistemi antena koji pokrivaju određenu geografsku oblast, pri čemu se signal do cilja prenosi preko niza antena.

- Zemaljski mikrotalasi - koriste antensku mrežu na Zemlji, a za komunikaciju koriste mikrotalase niske frekvencije koji zahtevaju da antene budu optički vidljive, pa se iste smeštaju na visoke tačke.

- Komunikacioni sateliti - koriste mikrotalase za komunikaciju tako što se prenos između dve tačke koje nemaju optičku vidljivost ostvaruje poprečnom komunikacijom preko satelita koji se nalaze u orbiti. Na ovaj

način se prenose televizijski i telefonijski signal. Brzina komunikacije je dosta mala $100Mbps$.

2.2.3 Mrežni softver

Mrežna infrastruktura sama po sebi ne služi ničemu bez mrežnog softvera. Uloga mrežnog softvera je da obezbedi korisniku mrežnu komunikaciju. Na primer, programer pregledača Veba, ne treba da misli o tome kako će pregledač primiti informacije, već treba da se fokusira samo na aspekte značajne za njegovu konkretnu aplikaciju, a sve ostale detalje prepusti nižem sloju mrežnog softvera.

Mrežni softver, najgrublje, može da se podeli na dva nivoa:

- niskog nivoa - mrežni softver koji omogućuje korišćenje različitih mrežnih uređaja, poput mrežnih kartica, modema, itd. Ovaj softver se nalazi u jezgru operativnog sistema i u obliku upravljača perifernim uređajima, takoznavnih drajvera (eng. driver). On upravlja računarskim hardverom i komunikacijskom opremom. Korisnik nikad ne koristi ovaj softver direktno, a često nije ni svestan njegovog postojanja
- visokog nivoa.

2.3 Literatura

Čitanje literature predstavlja neizostavan deo gradiva, kako naredni izvori predstavljaju odlican izvor informacija, na vama je da ih procitate:

- Predrag Janičić, Programiranje 1, Matematički fakultet, glava 1
- Filip Marić, Uvod u web i internet tehnologije, Matematički fakultet, glave 1 i 2
- Ajzenhamer Nikola, Bukurov Anja, Stanković Vojislav, Programiranje za Veb skripta, glave 1,2 i 3

3 HTML i CSS

HTML i CSS predstavljanju neizostavan materijal pri učenju web programiranja. Upravo zbog njihovog velikog značaja im posvećujem celo poglavlje. Slajdovi korišćeni prilikom predavanja su apsolutno nedovoljan materijal. Svi materijali korišćeni pri kreiranju časova, kao i ovog materijala su javno dostupni i nalaze se na:

- Aleksandar Veljković, Veb programiranje, Matematički fakultet http://poincare.matf.bg.ac.rs/ aleksandar/files/web/skripta.pdf
- W3Schools: https://www.w3schools.com/
- Filip Marić, Uvod u Veb i Internet tehnologije, Matematički fakultet

3.1 HTML

3.1.1 Zadaci sa časa

Neki od zadataka rađenih na času su navedeni u nastavku.

Primer 3.1 *Prva HTML stranica: Treba kreirati svoju prvu HTML stranicu korišćenjem tagova. Propozicije stranice su sledeće:*

- *Kreirati naslov stranice i iskoristiti barem 3 nivoa heading-a*
- *Napisati neki pasus koji ima smisla, u okviru kog treba iskoristiti break, strong i em tagove*
- *Potom kreirati isto to za ostale nivoe headinga*
- *Kreirati jos barem 3 stranice, od čega: 2 treba da sadrže smisleni tekst u paragrafima, a poslednja može sadržati tekst u vidu lorem ipsum dolor sit...*
- *Druga strana treba da sadrži linkove ka barem 3 spoljašnje strane, npr: ujutru uz kafu volim da čitam, pa linkove ka nekoliko portala*
- *Svaka strana treba da sadrži veze ka svim ostalim stranama*

Primer 3.2 *Španska kuhinja: Kreirati sajt prema sledećim propozicijama:*

1. *Iz kojih jela se sastoje tipični obroci u Španiji (entrada, primer plato, segundo plato,...)?*
2. *Kreirati osnovnu stranicu sa opisom delova obroka, a potom za svaki deo obroka kreirati posebnu stranicu.*
3. *Svaka stranica mora da sadrži*
    - *Naslov, npr.: Primer plato*
    - *Podnaslov, naziv nekog tipičnog jela karakterističnog za taj obrok, npr.: Nachos (za entradas)*
    - *Pasus, nešto o tom jelu, odakle je poteklo, kad je nastalo i slično*
    - *Pasus sa neuređenom listom potrebnih sastojaka*

- *Pasus sa uređenom listom postupka pripreme*

4. *Nekoliko slika jela*
5. *Vezu ka prethodnoj i narednoj stranici*
6. *Poslednja stranica treba da se vraća na prvu i da ima spoljašnju vezu ka opisu nekog grada u Španiji.*

Primer 3.3 *Sportski izveštaj: Kreirati stranicu o sportu. Odabrati sport po izboru, a potom ispuniti specifikaciju:*

1. *Napraviti definicionu listu koja opisuje odabrani sport*
2. *Napraviti listu sa pravilima igre, ako je moguće kreirati listu u listi (ugnježdena lista)*
3. *Kreirati tabelu sa rezultatima nekog takmičenja*
4. *Ubaciti dve ili više slika (iskoristiti width i height kako bi se veličina slike prilagodila)*
5. *Ubaciti tabelu sa rezultatima sa nekog većeg takmičenja iz tog sporta*

Primer 3.4 *Stiven Hoking: Kreirati sajt koji će sadržati informacije o Stivenu Hokingu (eng. Stephen Hawking). Specifikacija sajta je sledeća:*

- *Sajt treba da sadrži naslov, kome će u tooltip-u stajati: engleski teoretski fizičar, kosmolog, autor i direktor istaživanja u Centru za teorijsku kosmologiju na Univerzitetu u Kembrdžu.*
- *Ispod naslova treba da stoji kratki paragraf o Stivenu Hokingu, paragraf treba da sadrži boldovan i italic tekst, kao i nešto što bi bilo highlightovano i precrtano*
- *Kreirati listu sa spiskom njegovih publikacija*
- *Ubaciti sliku, i skalirati je korišćenjem odgovarajućih atributa*
- *Ubaciti neki citat na odgovarajući način*
- *Ubaciti citat nečega sa njegove stranice, i potom referisati stranicu.*
- *Ubaciti vezu ka drugoj stranici na kojoj će biti slike i nazivi nekih od njegovih publikacija*

Primer 3.5 *Forma za registraciju za učešće na nekom kursu. Zadatak je kreirati stranicu koja će sadržati formu, koja bi trebalo da se popuni kako bi se prijavilo za učešće na nekom od kurseva. Sami odaberite nazive kurseva, kao i relevantne informacije polaznika.*

- *Ime*
- *Prezime*
- *Adresa*

- *Grad*
- *Broj telefona*
- *Broj mobilnog*
- *e-mail adresa*
- *kurs za koji se prijavljuje - lista od barem 5 izbora*
- *odabir termina: vikendom, radnim danima ili svejedno - radio buttoni*
- *checklista: radim na svom računaru/ potreban mi je računar*
- *text area u kojoj treba upisati prethodno iskustvo*
- *button za slanje informacija*

Neke od informacija o polaznicima kurseva su obavezne, neke nisu, sami odredite koje jesu. Osim navedenih treba dodati jos barem 2 dodatne informacije po izboru u različitim oblicima.

3.2 CSS

3.2.1 Zadaci sa casa

Primer 3.6 *Kreirati stranicu zdrave hrane.*

3.2.2 Domaći zadaci

Domaći zadaci vezani za ovo poglavlje se odnose na unapređivanje i dodavanje sadržaja i isprobavanje tagova i elemenata nad zadacima rađenim na časovima.

4 JavaScript

4.1 Uvod

4.1.1 Obnavaljanje osnovnih konstrukta

Osnovni elementi za opis izračunavanja u programima nazivaju se naredbe. Naredbe za kontrolu toka omogućavaju različite načine izvršavanja programa, u zavisnosti od vrednosti promenljivih. Naredbe za kontrolu toka mogu biti:

- naredbe grananja i
- petlje.

Osnovni oblik naredbe koji se javlja je takozvana naredba izraza (ova vrsta naredbi obuhvata i naredbu dodele i naredbu poziva funkcije). Naime, svaki izraz završen karakterom ; je naredba. Na primer:

```
3 + 4*5;
n = 3;
c++;
f();
```

Nekada, želimo da više različitih naredbi grupišemo i da ih tretiramo kao jednu jedinstvenu naredbu. Vitičaste zagrade i se koriste da grupišu naredbe u složene naredbe, odnosno blokove, i takvi blokovi se mogu koristiti na svim mestima gde se mogu koristiti i pojedinačne naredbe.

Naredbe grananja (ili naredbe uslova), na osnovu vrednosti nekog izraza, odreduju naredbu (ili grupu naredbi) koja će biti izvršena.

```
if (izraz)
naredba1
else
naredba2
```

Naredbe *naredba*1 i *naredba*2 su ili pojedinačne naredbe (kada se završavaju simbolom ;) ili blokovi naredbi zapisani između vitičastih zagrada (na kraju kojih ne ide ;). Deo naredbe else je opcioni, odnosno, ne mora postojati, pa se može napisati samo if grana. Izraz izraz predstavlja logički uslov:

```
if (5 > 7)
   a = 1;
else
   a = 2;


if (7)
    a = 1;
else
    a = 2;

a = 3;
```

```
if (a = 0)
    console.log("a je nula\n");
else
    console.log("a nije nula\n");
```

Često možemo imati višestruke odluke, za šta koristimo *else if* konstrukciju oblika:

```
if (izraz1)
    naredba1
else if (izraz2)
    naredba2
else if (izraz3)
    naredba3
else
    naredba4
```

Primer 4.1 *Primer else if naredbe.*

```
if (a > 20)
   console.log("A je vece od 20\n");
else if (a > 10)
   console.log("A je vece od 10\n");
else if (a < -20)
   console.log("A je manje od -20\n");
else if (a < -10)
   console.log("A je manje od -10\n");
else
   console.log("A pripada intervalu [-10, 10]\n");
```

Ternarni operator je uslovni operator oblika: *uslov* ? *ispunjen* : *inace* i ima isto značenje kao i if else.

Primer 4.2 *Zadatak je smestiti veći od dva broja u promenljivu x. Prvi if else ima isto značenje kao i ternarni operator koji smešta veći broj u x. Ako je a vece od b postavlja se x na a, inače postavlja se na b.*

```
if (a > b)
   x = a;
else
   x = b;
x = (a > b) ? a : b;
```

Osim if else postoji i naredba switch, koja se takođe može koristiti za višestruko odlučivanje i ima opšti oblik:

```
switch (izraz) {
  case konstantan_izraz1: naredbe1
  case konstantan_izraz2: naredbe2
  ...
  default: naredbe_n
}
```

Case predstavljaju slučajeve, koji ako su ispunjeni, izvršava se naredba desno od dvotačke. U slučaju da nijedan od case-ova nije ispunjen izvršava se default, ako default nije postavljen i nijedan od case-ova nije ispunjen kroz switch će se samo proći.

Petlje (ciklusi ili repetitivne naredbe) uzrokuju da se određena naredba (ili grupa naredbi) izvršava više puta (sve dok je neki logički uslov ispunjen).

```
while(izraz)
   naredba

while (i < j)
   i++;

while (1)
   i++;

for (izraz1; izraz2; izraz3)
   naredba

izraz1;
while (izraz2) {
   naredba
   izraz3;
}

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

Postoji još jedan vid petlje, to je petlja do-while.

```
do {
naredbe
} while(izraz)
```

Telo (blok naredbi naredbe) naveden izmedu vitičastih zagrada se izvršava i onda se izračunava uslov (izraz *izraz*). Ako je on tačan, telo se izvršava ponovo i to se nastavlja sve dok izraz *izraz* nema vrednost nula (tj. sve dok njegova istinitosna vrednost ne postane netačno). Za razliku od petlje while, naredbe u bloku ove petlje se uvek izvršavaju barem jednom.

U nekim situacijama pogodno je napustiti petlju ne zbog toga što nije ispunjen uslov petlje, već iz nekog drugog razloga. To je moguće postici naredbom break:

```
for(i = 1; i < n; i++) {
    if(i > 10)
       break;
    ...
}
```

Naredbom continue se prelazi na sledeću iteraciju u petlji. Na primer,

```
for(i = 0; i < n; i++) {
   if (i % 10 == 0)
      continue; /* preskoci brojeve deljive sa 10 */
   ...
}
```

Petlje i uslovne naredbe se mogu kombinovati, korišćenjem jednih u drugima i slično. U slučaju ugnježdenih petlji, naredbe break i continue imaju dejstvo samo na unutrašnju petlju.

Funkcije Funkcija je jedna od osnovnih konstrukcija jezika koja nam omogućava ponovno korišćenje koda. Omogućavaju nam čak i da ih koristimo iako ne razumemo detalje implementacije. Izdvajanjem koda u funkcije povećava se čitljivost koda i olakšava se njegovo održavanje.
Osnovno uputstvo za pisanje funkcija se sastoji iz nekoliko bazičnih koraka:

- Uočiti logičke celine
- Uočiti neophodne parametre i povratnu vrednost
- Dati ime koje odgovara implementaciji
- Implementirati funkciju

Funkcija se odlikuje svojom deklaracijom i definicijom. Deklaracija (ili prototip) funkcije ima opšti oblik:
$tip\ ime\_funkcije(niz\_deklaracija\_parametara)$;

Definicija funkcije ima oblik:

```
tip ime_funkcije(niz_deklaracija_parametara) {
   deklaracije
   naredbe
}
```

Funkcija može imati parametre koje obraduje i oni se navode u okviru definicije, iza imena funkcije i izmedu zagrada. Termini parametar funkcije i argument funkcije se ponekad koriste kao sinonimi. n je parametar funkcije kvadrat(n);, a 5 i 9 su njeni argumenti u pozivima kvadrat(5) i kvadrat(9). Parametri funkcije mogu se u telu funkcije koristiti kao lokalne promenljive te funkcije a koje imaju početnu vrednost odredenu vrednostima argumenata u pozivu funkcije.

Funkcija rezultat vraća naredbom return r; gde je r izraz zadatog tipa ili tipa koji se može konvertovati u taj tip. Naredba return r; ne samo da vraća vrednost r kao rezultat rada funkcije, nego i prekida njeno izvršavanje.

Funkcije mogu pozivati druge funkcije, a funkcija može pozivati i samu sebe.

4.2 Zadaci sa casa

4.3 Domaći zadaci

Primer 4.3 *Na koje if se odnosi else?*

```
if (izraz1)
    if (izraz2)
        naredba1
else
    naredba2
```

Primer 4.4 *Na koje if se odnosi else?*

```
if (izraz1) {
   if (izraz2)
      naredba1
} else
    naredba2
```

Primer 4.5 *Napisati program koji za dva cela broja ispisuje najpre njihove vrednosti, a zatim i njihov zbir, razliku, proizvod, ceo deo pri deljenju prvog broja drugim brojem i ostatak pri deljenju prvog broja drugim brojem.*

Primer 4.6 *Napisati program koji za uneti pozitivan trocifreni broj na standardni izlaz ispisuje njegove cifre jedinica, desetica i stotina*

Primer 4.7 *Napisati program koji za uneti pozitivan četvorocifreni broj:*
(a) izračunava proizvod cifara
(b) izračunava razliku sume krajnjih i srednjih cifara
(c) izračunava sumu kvadrata cifara
(d) izračunava broj koji se dobija ispisom cifara u obrnutom poretku
(e) izračunava broj koji se dobija zamenom cifre jedinice i cifre stotine

Primer 4.8 *Napisati program koji za realne vrednosti dužina stranica pravougaonika ispisuje njegov obim i površinu. Ispisati tražene vrednosti zaokružene na dve decimale.*

Primer 4.9 *Napisati program koji za tri cela broja ispisuje njihovu artimetičku sredinu zaokruženu na dve decimale.*

Primer 4.10 *Napisati program koji za dva cela broja a i b dodeljuje promenljivoj rezultat vrednost 1 ako važi uslov: a) a i b su različiti brojevi b) a i b su parni brojevi c) a i b su pozitivni brojevi, ne veći od 100 U suprotnom, promenljivoj rezultat dodeliti vrednost 0. Ispisati vrednost promenljive rezultat.*

Primer 4.11 *Napisati program koji za dva cela broja ispisuje njihov maksimum.*

Primer 4.12 *Napisati program koji za dva cela broja ispisuje njihov minimum.*

Primer 4.13 *Napisati program koji za tri cela broja ispisuje zbir pozitivnih.*

Primer 4.14 *U prodavnici je organizovana akcija da svaki kupac dobije najjeftiniji od tri artikla za jedan dinar. Napisati program koji za cene tri artikla izračunava ukupnu cenu, kao i koliko dinara se uštedi zahvaljujući popustu.*

Primer 4.15 *Broj je Armstrongov ako je jednak zbiru kubova svojih cifara. Napisati program koji za dati trocifren broj proverava da li je Armstrongov.*

Primer 4.16 *Napisati program koji ispisuje proizvod parnih cifara unetog četvorocifrenog broja.*

Primer 4.17 *Napisati program koji za ceo broj x ispisuje njegov znak, tj da li je broj jednak nuli, manji od nule ili veći od nule.*

Primer 4.18 *Napisati program koji za redni broj dana u nedelji ispisuje ime odgovarajućeg dana. U slučaju pogrešnog unosa ispisati odgovarajuću poruku*

Primer 4.19 *Napisati program koji za uneti karakter ispituje da li je samoglasnik.*

Primer 4.20 *Napisati program koji 5 puta ispisuje tekst Mi volimo da programiramo.*

Primer 4.21 *Napisati program koji preko prompta učitava ceo broj n i ispisuje n puta tekst Mi volimo da programiramo.*

Primer 4.22 *Napisati program koji učitava pozitivan ceo broj n a potom ispisuje sve cele brojeve od 0 do n.*

Primer 4.23 *Napisati program koji učitava dva cela broja n i m ispisuje sve cele brojeve iz intervala [n, m].*
(a) Koristiti while petlju.
(b) Koristiti for petlju.
(c) Koristiti do-while petlju.

Primer 4.24 *Napisati program koji učitava ceo pozitivan broj i izračunava njegov faktorijal.*

Primer 4.25 *Preko prompta unose se realan broj x i ceo pozitivan broj n. Napisati program koji izračunava n-ti stepen broja x.*

Primer 4.26 *Pravi delioci celog broja su svi delioci sem jedinice i samog tog broja. Napisati program za ceo pozitivan broj x ispisuje sve njegove prave delioce.*

Primer 4.27 *Napisati program koji za uneti prirodan broj ispisuje da li je on deljiv sumom svojih cifara.*

4.4 Dodatni zadaci

Primer 4.28 *Kreirati sajt zdrave hrane. Koristeći ugrađenu funkciju prompt zahtevati od korisnika količinu badema koju želi da kupi(u gramima), a potom u alert prozoru ispisati konačnu cenu za unetu količinu. Npr. ako je badem 2000 dinara po kilogramu, a korisnik želi 500 grama, u alertu treba da se ispiše 1000 dinara.*

5 Zaključak

A Dodatak