J.U. MJEŠOVITA SREDNJA ELEKTROTEHNIČKA ŠKOLA TUZLA

prof. Imširović Edin
prof. Bojić Dejan

INFORMATIKA II

Interna skripta za učenike drugog razreda
2. polugodište

Tuzla, januar 2011. god.

SADRŽAJ

1. RAČUNARSKA GRAFIKA

1.0. UVOD

Grafika je vizuelna prezentacija informacija pomoću slika tj. boja i oblika na nekoj podlozi, kao što je npr. zid, platno, računarski ekran, papir ili kamen, a cilj joj je da informira, ilustrira ili zabavi gledaoca ili čitaoca. Primjeri su: fotografije, crteži, grafikoni, dijagrami, tipografija, brojevi, simboli, geometrijski oblici, karte ili slično. To su vizuelni elementi koji se često koriste kao zamjena za tekst da pomognu čitatelju u lakšem razumijevanju nekog koncepta ili da koncept učine zanimljivim, kombinirajući pri tome tekst, ilustraciju i boju.

Računarska grafika je grafika kreirana pomoću računara i, uopšteno, obuhvata stvaranje, pohranjivanje i obradu slikovnog sadržaja pomoću računara. Razvoj računarske grafike (engl. *Computer Graphics* – CG) započinje sa pojavom personalnih računara. Visoko razvijena sposobnost prepoznavanja oblika kod čovjeka čini računarsku grafiku jednim od najprirodnijih načina komunikacije sa računarom. Grafička interakcija između čovjeka i mašine postala je standardnim sastavnim dijelom računarskih korisničkih interfejsa. Danas gotovo svi računari koriste neku vrstu grafike i korisnicima je omogućeno korištenje ikona i slika umjesto same tastature.

U današnje vrijeme računara i računarski stvorena grafika ulaze u mnoge aspekte naše svakodnevnice. Nalazimo ih u televiziji, novinama, vremenskim izvješćima ili tijekom operacijskih zahvata. Obično, pojam **računarske grafike** ima nekoliko značenja:

- reprezentacija i manipulacija nad slikovnim podacima pomoću računara
- različite tehnologije koje se koriste za stvaranje i manipulaciju takvih slikovnih podataka
- slike proizvedene uz pomoć računara
- potkategorija računarskih znanosti koja proučava metode za digitalnu manipulaciju i generisanje vizualnog sadržaja

1.1. POVIJEST RAČUNARSKE GRAFIKE

Izraz "Računarska grafika" prvi je upotrijebio 1960. godine William Fetter, grafički dizajner za Boeing kako bi opisao svoje istraživanje na računarskoj simulaciji ljudske figure koja vjerno simulira ljudsko tijelo u raznim fizičkim okruženjima.

Računarska grafika se razvila nakon pojave hardvera za računarsku grafiku. Rani projekti kao *Whirlwind* i *SAGE Projects* su potpomogli razvoju računarsko grafičke discipline. Daljnji napredak u razvoju digitalnih računarara kao što je TX-2 računar iz 1959. godine razvijen na Lincoln Laboratory-u doveo je do daljnjeg razvoja u interaktivnoj računalnoj grafici. To je prvi interaktivni računarski grafički sistem koji je ima potpuno definisani interfejs između čovjeka i mašine. Kao izlazni uređaj koristio je monitor a svjetleće pero putem *Sketchpad* programa, kao ulazni uređaj.

Godine 1969. ACM je inicirao SIGGRAPH (engl. *A Special Interest Group in Graphics*) posvećenu simulaciji i modeliranju, uređivanju teksta i kompoziciji, računalno generiranoj umjetnosti, kartografiji i mapiranju, računalno potpomognutom dizajnu, te softveru i hardveru računarske grafike.

Tokom 1970-tih personalni računari postaju sve moćnije grafičke alatke, sposobne crtati jednostavne i složene oblike i dizajne. 1980-tih umjetnici i grafički dizajneri počeli su na personalne računare, posebno na Commodore, Amiga i Macintosh, gledati kao ozbiljan dizajnerski alat, koji može uštedjeti vrijeme i crtati preciznije od drugih tada dostupnih metoda.

3D računarska grafika postala je moguća krajem 1980. godina pojavom moćnih SGI računara, koji su kasnije korišteni za izradu prvih računalno-generiranih kratkih filmova u Pixaru. Macintosh je ostao i dalje jedan od najpopularnijih alata za računarsku grafiku u studijima i poslovima grafičkog dizajna.

Moderni računarski sistemi koji datiraju od 1980-tih na dalje, često koriste grafičko korisničko sučelje (engl. *Graphics User Interface* - GUI) za prezentaciju podataka i informacija pomoću simbola, ikona i slika, radije nego tekstom. Grafika je jedan od 5 ključnih elemenata multimedijske tehnologije.

Računarska grafika je postala popularna 1990-tih u igrama, multimediji i animaciji. Jedna od prvih cjelovitih 3D igara – *Quake* izdana je 1996. godine a samo godinu ranije 1995., prvi dugometražni računarsko generirani animirani film – *Toy Story* prikazan je u kinima širom svijeta. Od tada, računarska grafika postala je preciznija i detaljnija, zbog naprednijih računara i boljih aplikacija za 3D modeliranje, kao što je *Cinema 4D*.

1.2. PRIMJERI PRIMJENE RAČUNARSKE GRAFIKE

Računarska grafika danas se koristi u različitim područjima privrede, administracije, edukacije, zabave i svakodnevnog kućnog života. Područje primjene se ubrzano širi s rasprostranjenošću računara. Neki primjeri primjene računarske grafike uključuju:

- **korisničke interfejse** – većina aplikacija na personalnim računarima i na radnim stanicama imaju grafički sistem prozora (GUI) pomoću koga komuniciraju sa korisnicima. Primjeri takvih aplikacija uključuju obradu teksta, stolno izdavaštvo, proračunske tabele, itd..
- **interaktivno crtanje** – u poslovnim, naučnim i tehnološkim primjenama računarska grafika koristi se za prikazivanje funkcija, dijagrama, histograma i sličnih grafičkih prikaza sa svrhom jasnijeg sagledavanja složenih pojava i olakšanja procesa odlučivanja
- **kancelarijska automatizacija i elektronsko izdavaštvo** – računarska grafika široko se koristi za izradu elektronskih i štampanih dokumenata
- **projektovanje pomoću računara** (engl. Computer Aided Design – CAD) - danas se standardno koristi za projektovanje sistema i komponenata u mašinstvu, elektrotehnici, elektronici, telekomunikacijama, računarstvu, itd..
- **simulacija i animacija** – računarska grafika koristi se za naučnu i inženjersku vizuelizaciju i zabavu; područja primjene obuhvataju prikaze apstraktnih matematičkih modela vremenski promjenljivih pojava, TV i filmsku tehnologiju, itd..
- **umjetnost** – računarska grafika se koristi za kreiranje umjetničkih slika – digitalna umjetnost
- **trgovina** – računarska grafika se koristi za vizuelnu animaciju i elektronsku trgovinu
- **upravljanje procesima** – podaci iz senzora dinamički se prikazuju u prikladnom grafičkom obliku
- **geografski informacioni sistemi** – računarska grafika koristi se za tačan prikaz geografski raspodijeljenih i rasprostranjenih sistema i mjernih podataka npr. u telekomunikacijama i telemetriji
- **grafičko programiranje** – računarska grafika se koristi za automatizaciju procesa programiranja virtuelnih sistema, npr. u instrumentaciji
- **video igre** – jedan od glavnih elemenata igara je računalska grafika koja se koristi za prikaz virtualnih prostora

1.3. VRSTE RAČUNARSKE GRAFIKE

Najjednostavnija definicija grafike je da je to prezentacija informacija pomoću slika tj. boja i oblika. Računarska grafika je isto to, s tim što se za generisanje i prezentaciju slikovne informacije koristi računar. Postoji više načina da se podjeli ono što nazivamo "Računarskom grafikom".

Prva osnovna podjela je na interaktivnu i ne interaktivnu grafiku. Interaktivna grafika podrazumijeva dinamičan način prikaza slike na mediju koji to omogućava (displej) i, preko odgovarajućeg interfejsa, aktivno učešće čovjeka (dizajnera) u stvaranju i izmjeni slike, pri čemu su rezultati odmah vidljivi. Ne interaktivnom računarskom grafikom smatra se svako generisanje ili prezentiranje slikovnih informacija koje ne zadovoljava prethodne uslove.

Mnogo raširenija i češće upotrebljavana podjela računarske grafike je podjela na ***vektorsku*** i ***rastersku*** grafiku. Ova podjela je izvršena prema osnovnim gradivnim elementima slike.

Kod **vektorske grafike**, gradivni elementi su objekti (prave i krive linije, otvoreni i zatvoreni, ispunjeni i neispunjeni geometrijski oblici) koji mogu da se preklapaju, prekrivaju ili uklapaju i tako tvore sliku. Raspored objekata se može mijenjati isto kao i njihov oblik i veličina a da pri tome položaj i karakteristike ostatka objekata na slici ostane nepromijenjen. Ovakve slike je lakše stvarati, mijenjati i kombinovati sa drugim slikama. Najadekvatnije ih je porediti sa kolažima od komadića raznobojnog papira. Računarska interna reprezentacija ovakvih slika je niz matematskih vektorskih formula koje opisuju način i redoslijed iscrtavanja objekata. Tome ova vrsta računarske grafike duguje ime. Vektorska grafika se često naziva i objektna grafika.

Vektorska grafika svoju primjenu nalazi u CAD programima, programima namijenjenim dizajnerima i svuda gdje je sastavljanje slike od objekata prirodan način vizualizacije stvarnog ili izmišljenog svijeta. Takođe, vektorskoj grafici nema alternative ako želimo pomoću računara simulirati trodimenzionalni svijet.

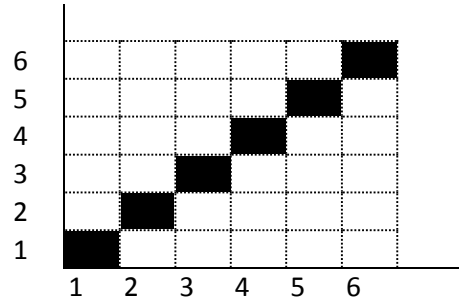
Za prikaz prizora koji se sastoje od jako mnogo detalja koji ne stoje u matematski opisivoj vezi (fotografije) vektorska grafika je potpuno nepodesna. U tim slučajevima se koristi rasterska grafika.

Rasterska grafika kao osnovni gradivni element slike koristi tzv. **piksel** (složenica od dvije engleske riječi: *Picture* i *Element*). Piksel je najmanji dio slike koji ima jedinstvene vrijednosti boje i/ili intenziteta osvijetljenosti. Kod nas se izraz piksel često prevodi kao tačka. Pojednostavljeno, rasterska slika je slika sastavljena od tačaka različitog nivoa osvijetljenosti (monohromatske slike) ili različitih boja (kolor slike). Više riječi o rasterima biće u nastavku teksta.

Još jedna podjela se često spominje, a to je na **dvodimenzionalnu** (**2D**) i **trodimenzionalnu** (**3D**) grafiku. Nije posebno potrebno objašnjavati šta je dvodimenzionalno a šta trodimenzionalno. Ipak, kada je računarska grafika u pitanju, uz ova dva pojma vezane su neke zabune i zablude. U 2D grafici moguće je pomoću osvjetljenja i sjenki ili pomoću boja i oblika dočarati trodimenzionalni svijet. Fotografije su, na primjer, dvodimenzionalne slike, ali vrlo vjerno prikazuju trodimenzionalni svijet. U nekim programima za rad sa tekstom postoji mogućnost prikaza ispupčenih slova. To ipak nije trodimenzionalna grafika jer bilježi trodimenzionalni svijet u jednom trenutku tj. u jednom položaju. Na takvoj slici ne možemo pogledati neki objekat iz drugog ugla. 3D grafika podrazumijeva da se "slike" sastoje od objekata u virtuelnom prostoru u memoriji računara. Slika koju ćemo vidjeti zavisi od prostornih odnosa između ovih objekata i od ugla posmatranja. U 3D grafici se zato ne govori o slikama nego o svjetovima. Naravno, svi danas široko rasprostranjeni uređaji za prezentaciju grafike stvorene računarom su dvodimenzionalni što znači da se i 3D svjetovi moraju prilagoditi dvodimenzionalnom prikazu.

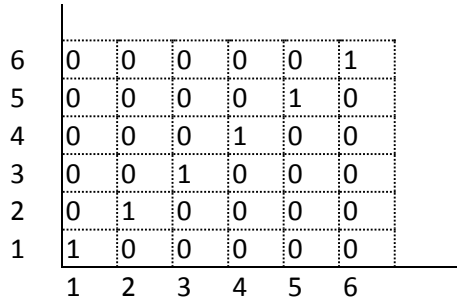
1.3.1. VEKTORSKA I BITMAP GRAFIKA

Razumijevanje razlike između programa za crtanje i slikanje, odnosno vektorske i bitmap grafike je osnova za shvaćanje rada grafičkih paketa. Razliku ćemo objasniti na jednom primjeru. Na slici 1. je prikazan koordinatni sistem sa po 6 jedinica na apcisi i ordinati. Jedan kvadratić predstavlja jedan *pixel* na ekranu (Pixel je skraćenica od "*picture element*").



Slika 1. Slika sastavljena od bijelih i crnih piksela

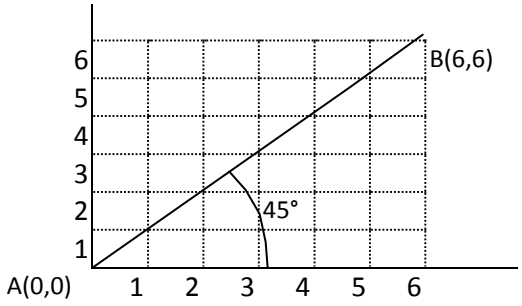
Želimo li ovu sliku predstaviti bitmap-om, tada to možemo učiniti interpretiranjem upaljene tačke "logičkom jedinicom", a ugašene "logičkom nulom". Nakon pretvorbe tačaka u bitove, sliku bi mogli predstaviti kao: 000001, 000010, 000100, 001000, 010000, 100000 ili dekadno 1,2,4,6,8,16,32. Kako svaku tačku možemo predstaviti jednim bitom, a kao što možemo vidjeti na slici 2 ti se bitovi nalaze u nekakvoj križaljci – mapi, ovu vrstu opisivanja zovemo bitmapa ili doslovno mapa bita (Bit je skraćenica od BInary digiT – binarna cifra).



Slika 2. Prezentacija piksela binarnim vrijednostima

Da je kojim slučajem slika u boji, morali bi za svaku tačku još upisati i podatak o boji, npr. otvoriti još dva bita za svaku tačku ako se radi o paleti od 8 boja (*Red, Green, Blue* – Crvena, Zelena i Plava). Po jedan bit za svaku osnovnu boju. U slučaju da se radi o bijeloj boji uključili bi smo sva tri bita, u slučaju da je uključena Plava i Zelena dobije se cijan itd.. Za zapisivanje ove potrebno je 6 kombinacija od 6 bita, ako se radi o monohromatskoj slici odnosno 18 kombinacija za slučaj slike u boji.

Kako bi smo ovu sliku opisali vektorski? Način opisivanja ćemo krajnje pojednostaviti, jer se način opisivanja krivulje znatno usložnjava. Veličine koje izražavamo pozitivnim ili negativnim brojevima nazivamo skalarnim veličinama (temperatura, zapremina, dužina, …), a veličine kod kojih se vrijednosti određuju, kako dimenzijama tako i smjerom u prostoru nazivamo vektorima (sila, brzina, itd…). Vektor je odsječak koji ima određenu dužinu i smjer. Možemo ga predstaviti u Dekartovom koordinatnom sistemu ili polarnom sistemu. Pretpostavimo da se radi o polarnom sistemu. Liniju ćemo opisati tako da kažemo u kojoj tački počinje, u kojoj završava i koliki je ugao naspram apcise. To bi mogli zapisati sa: 0,0,6,6,45. Možemo primijetiti da je zapis kraći nego kod bitmape. Želimo li liniju obojiti potrebna je još jedna informacija, ali svakom elementarnom dijelu linije ne mogu se davati individualni opisi boja.



Slika 3. Vektorski zapis slike

Zapis bitmape je jednostavniji i duži dok je zapis vektorom složeniji kraći. Nedostatak bitmape je da se u slučaju izmjene veličine slike dolazi do znatnog izobličenja. Kod vektorske grafike izobličenja praktično nema jer se na osnovu matematičke definicije slike (linija, krivih, poligona, 3D objekata…) slika je uvijek ista ili su izobličenja neznatna.

Današnji savremeni grafički paketi kao što su *CorelDRAW*, *Photoshop*, *Daneba Canavas*, *Xara*, itd.., u sebi objedinjuju vektorsku i bitmap grafiku.

1.4. DVODIMENZIONALNA RAČUNARSKA GRAFIKA

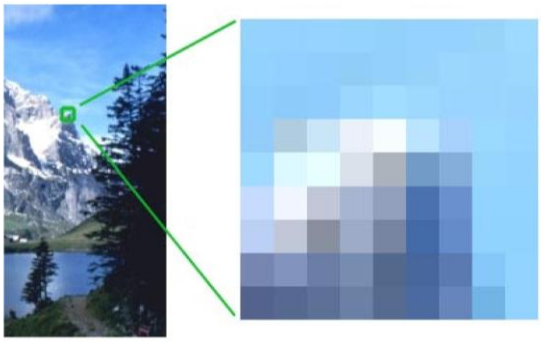
Dvodimenzionalna računarska grafika je računarski bazirana generacija digitalnih slika – većinom 2D modela (2D objekti, tekst, slike), koja se uglavnom koristi u aplikacijama namijenjenim standardnim tehnikama crtanja i štampanja kao što je tipografija, kartografija, tehničko crtanje, reklame itd..

2D grafički modeli mogu kombinirati geometrijske modele (kod vektorske grafike), digitalne slike (kod rasterske grafike), tekst za prikaz (sa svojim osobinama: font, stil, veličina, boja), matematičke funkcije i jednačine i sl.. Ove komponente mogu se modifikovati i njima manipulisati putem geometrijskih transformacija kao što je translacija i rotacija, skaliranje i slično, kako bi zauzele odgovaraju poziciji i orijentaciju unutar cjelokupnog modela.

1.4.1. RASTERSKE SLIKE (BITMAPE)

Rasteri su slike sastavljene od pravilno raspoređenih elemenata fiksne veličine i oblika za koje je određena boja ili tonalitet (nivo osvijetljenosti). Ovi elementi nazivaju se *pikseli*.

Ako su pikseli dovoljno mali i dovoljno gusto postavljeni, ljudsko oko ih ne primjećuje kao zasebne elemente već boju ili tonalitet pojedinačnih piksela veže u kontinualnu sliku. Ova osobina ljudskog oka naziva se (na engleskom) ***SPATIAL INTEGRATION*** i ima veliki značaj za vizualizaciju, ne samo u računarskoj grafici.



Slika 4. Primjer rasterske slike

Sa lijeve strane je cijela slika, a desno se nalazi njezin dio na kome je prikazan jedan od vrhova planine, uveličan za 250 procenata. Očigledno je, da se slika sastoji od matrice (redova i kolona) malih elemenata različitih boja. Takvi elementi nazivaju se pikseli. Ova matrica (pravougaona mreža) piksela naziva se još i **raster** odakle ovaj vid grafičkog zapisa slike i nosi naziv.

Ljudsko oko ne razlikuje pojedinačne elemente (jer su suviše sitni), tako da mi vidimo cijelu sliku sa blagim prelazima između boja. Da bi se rasterska slika opisala i zapamtila na digitalnom mediju i da bi se obrađivala pomoću PC-a, potrebno je za svaki piksel koji čini raster znati poziciju i vrijednost boje ili tona.

Pojednostavljeno, raster se može zamisliti kao dvodimenzionalna mreža postavljena u koordinatni sistem u čijem se svakom čvoru nalazi po jedan piksel. Ovaj koordinatni sistem može biti pravougaoni, ali to nije obavezno. Brojeći čvorove od koordinatnog početka do odabranog piksela, po obadvije ose, svakom pikselu se može dodijeliti jedinstvena adresa. Znajući adresu piksela i oblik mreže (koordinatnog sistema) lako određujemo poziciju bilo kojeg piksela u rasteru.

Ako se uspostavi sistem preslikavanja iz svijeta boja u svijet brojeva, tada se svakoj boji kojom piksel može biti obojen može dodijeliti jedinstven broj. Zavisno od broja boja koje se žele prikazati, potrebno je za svaki piksel odvojiti manji ili veći memorijski prostor. Jednim bitom informacije mogu se opisati dva stanja (dvije boje). Ako želimo prikazati veći broj boja, moramo povećati broj bita kojima se opisuje boja piksela.

Ovo međutim povećava zahtjeve za memorijskim prostorom za smještaj takvih slika i za procesorskom snagom i propusnošću računara na kojima se slike obrađuju. Zato su danas standardno u upotrebi formati rasterskih slika sa 1 (2 boje), 4 (16 boja), 8 (256 boja), 16 (64k boja), 24 (16M boja) ili 32 (4G boja) bita po pikselu za opis boje, a samo u profesionalnim primjenama i sa 36, 40 ili više bita.

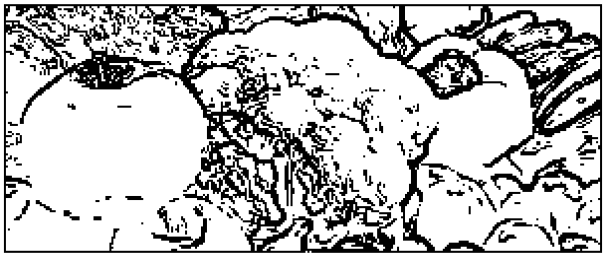
Ovo je bitna veličina za svaki raster i često se naziva **DUBINA BOJE** i izražava se ili u broju bita za opis boje ili u broju boja koje se mogu prikazati.

1.4.1.1. TIPOVI RASTERSKIH SLIKA

Rasterske slike mogu da sadrže bilo koji broj boja, ali se po tome najčešće dijele na četiri osnovne kategorije:

Monohromatski rasteri (*bitmaps*)

Naziv za slike koje se sastoje od samo dvije boje. Obično se koriste crna i bijela boja, ali moguća je i kombinacija bilo koje druge dvije boje. Ponekad se za takve slike koristi naziv bitmapa (engl. *bitmap*), zato što računar koristi samo jedan bit za svaki piksel[1].



Slika 5. Monohromatski raster

***Grayscale* rasteri**

To su rasteri u kojim svaki piksel može primiti bilo koju boju iz skale sivih nijansi, od crne do bijele boje. Realističnost *grayscale* slika zavisi od glatkoće prelaza između površina obojenih susjednim nijansama sive boje i to za cijelu skalu sivih tonova koji se mogu prikazati. Ovo opet zavisi od broja sivih tonova u skali između crne i bijele boje kao i od njihove raspodjele u toj skali.

Danas se najčešće koristi skala od 256 sivih tonova (računajući i bijelu i crnu boju). Za smještanje 256 nijansi sive boje potrebno je obezbjediti po 8 bita (1 bajt) za svaki piksel u rasteru. Ovakvi rasteri, uz dobru distribuciju sivih tonova u skali, mogu obezbjediti gotovo potpuno gladak prijelaz iz crne u bijelu boju (koliko to ljudsko oko može registrovati).

Slika 6. Grayscale raster

Za primjene u profesionalnoj računarskoj obradi fotografija, 256 nijansi sive boje često nije dovoljno pa se koriste i rasteri sa 512 ili 1024 sive nijanse, odnosno 9 ili 10 bita po svakom pikselu u rasteru.

Višebojni rasteri

Takve slike sadrže nijanse dvije ili više boja. Najčešće se koriste takozvani duotonovi, koji se obično sastoje od crne i neke druge boje (obično *Pantone*). Na primjer, slika dole je sastavljena od crne i *Pantone Warm Red*.

[1] U tom slučaju, ukoliko se koristi više bita za svaki piksel naziv je *pixmap*.

Slika 7. Primjer višebojnog rastera

Kolor rasteri

Ako svaki piksel u rasteru može primiti bilo koju boju iz na neki način definisanog skupa boja (termin koji se najčešće koristi je paleta boja), tada govorimo o kolor rasteru. Kako se definiše ova paleta boja zavisi od izbora kolor modela i dubine boja rastera. Bez obzira na izbor kolor modela, svaka boja se sastoji od nekoliko komponenti, obično 3 (RGB, CMY, HSB) ili 4 (CMYK). Ove komponente se često nazivaju "kanali". Za početak ćemo pretpostaviti da se radi o RGB kolor modelu, mada je principijelno isto za bilo koji drugi model.

Svaka boja u RGB modelu sastoji se od tri komponente, crvene, zelene i plave. Boju piksela možemo zapisati pomoću tri broja, za svaku komponentu boje po jedan. Za svaku komponentnu boju, ovaj broj se kreće između broja koji označava minimalnu i broja koji označava maksimalnu vrijednost obojenosti piksela tom komponentnom bojom. Ako, na primjer, za opis obojenosti piksela svakom od komponentnih boja odvojimo po jedan bit informacije, tada svaka komponentna boja može biti "uključena" ili "isključena", što nam na raspolaganje stavlja paletu od 8 boja. Ovo može biti dovoljno ako želimo prikazati neki jednostavan crtež, ali za prikaz kolor fotografije, na primjer, potpuno je neadekvatno.

Slika 8. Kolor raster

Današnji grafički uređaji i formati zapisa kolor rastera uglavnom podržavaju 256 nivoa po kanalu, tj. za svaku komponentu boje dozvoljavaju 8 bita informacije po pikselu. Za RGB kolor model ovo znači 24 bita informacije o boji po pikselu, što omogućava prikaz više od 16.6 miliona boja.

Ovakav način formiranja palete boja je najjednostavniji ali nije uvijek primjenjiv. Prije svega, rasteri sa 24-bitnom bojom su veoma memorijski zahtjevni (lako je izračunati, raster veličine 1024×1024 piksela u 24 bitnoj grafici zahtjeva 3MB memorijskog prostora, a pri rezoluciji od 300 dpi to je sličica 8×8 cm). Osim toga, neki grafički uređaji ne podržavaju rad sa rasterima u 24-bitnoj boji (naročito u bližoj ili daljoj prošlosti kada je postavljena većina i danas važećih standarda).

1.4.1.2. KARAKTERISTIKE RASTERSKIH SLIKA

Slika se interpretira kao skup piksela date osvjetljenosti ili boje. Rasterski podaci zauzimaju mnogo mjesta (velike datoteke) ali koristeći se različitim vrstama kompresije ta se veličina može znatno smanjiti.

Ako pokušamo da uvećamo rastersku sliku primijetiti ćemo njihov osnovni nedostatak: ako je previše uveličamo, slika će izgledati neprirodno i početi će da se raspada na pojedinačne elemente. Iako je to manje primjetno, veliko smanjivanje slike takođe dovodi do gubitka kvaliteta. Objekti (ako postoje) su skupovi piksela tj. nema semantike objekata („slovo", „kružnica", „crta") što znači da ne postoji mogućnost izdvajanja pojedinih objekata unutar slike.

Kvalitet slike zavisi od rezolucije u trenutku kreiranja, dok slike izgledaju „prirodnije" od vektorskih.

Raster se lako štampa, sve dok štampač ima dovoljno memorije. Svi novi uređaji ne stvaraju probleme, ali kod ogromnih datoteka štampanje zna da potraje i kod najnovijih mašina.

1.4.1.3. KOMPRESIJA RASTERSKE SLIKE

Jedan od osnovnih nedostataka rasterske grafike jeste veličina datoteka u kojima su pohranjene rasterske slike naročito one velike rezolucije. Da bi se djelimično smanjio potrebni fizički prostor za pohranu rasterskih slika uvode se različite metode za smanjenje veličine (kompresije) digitalnih slika. Prilikom kompresije ne mijenja se broj piksela koji tvore sliku, samo se mijenja način na koji se slika priprema za pohranu, pri čemu kvaliteta slike direktno ovisi o stepenu i načinu kompresije.

Za razliku od binarnih podataka kod kojih je potrebno da dekompresirani podaci budu identični originalu, kod slika moguće su dvije vrste kompresije:

- bez gubitka kvalitete – *lossless compression* (TIFF, GIF, PNG i dr.)
- sa gubitkom kvalitete – *lossy compression* (JPEG)

Razlika između navedenih metoda jeste u stvari u prirodi ljudskog čula vida gdje se njegove osobine koriste kod *lossy* algoritama kompresije koji tolerišu određeni gubitak izvornih podataka prilikom postupka kompresije. Kompresija bez gubitaka se prvenstveno preporučuje kod medicinskih slika, tehničkih crteža, ikona i sl.. S druge strane *lossy* metode kompresije su pogodne za prirodne slike poput fotografija.

Kompresija se temelji na sličnosti susjednih područja na slici. Pojednostavljeno rečeno, pri zapisu informacija o sadržaju slike tada nije nužno pamtiti stvarne vrijednosti nego samo razlike u odnosu na susjedno područje. Uštede pri takvom načinu zapisa nisu male. Međutim, koristeći činjenicu da ljudsko oko nije savršeno, algoritmi za kompresiju slikovnih podataka ponekad vrlo agresivno obrađuju sliku.

Nova slika može biti i nekoliko desetaka puta manja od originalne. Negativna strana takvog pristupa je određeni gubitak informacija o slici. Jedan od najpoznatijih formata *lossy* kompresije fotografija je JPEG (engl. *Joint Photographic Experts Group*) format.

a) Originalna JPEG fotografija **b) Komprimirana JPEG kopija (najlošiji kvalitet)**

Slika 9. Razlika u kvaliteti prikaza s obzirom na stepen kompresije

Loša strana kod JPEG formata je ta što se prilikom kompresije javlja određeni gubitak kvalitete ovisno o stepenu kompresije (Slika 9.a. i 9.b.). Što je veća kompresija, slika zauzima manje memorije, ali zato je kvaliteta slike lošiji. Ovaj format je dizajniran posebno za digitalnu fotografiju, zauzima relativno malo memorijskog prostora i zato je pogodan za Web, slanje preko *e-maila* i slično. Preporučljivo je uvijek izabrati najmanji stepen kompresije, a kada se slika obrađuje na računaru, ako je to potrebno (npr. za stavljanje slika na Internet), povećati kompresiju slike.

Pored JPEG formata u digitalnoj fotografiji koristi se i TIFF (engl. *Tagged Image File Format*), industrijski standard slika dizajniran za objavljivanje fotografija u novinama i časopisima. TIFF koristi *lossless* kompresiju što znači da postoji kompresija (smanjuje se veličina) slike, ali ne gubi se na kvaliteti, niti na podacima slike. Loša strana ovog formata je mala kompresija, tj. slike ipak zauzimaju dosta prostora.

1.4.1.4. SLOJEVI (LAYERS)

Modeli koji se koriste u 2D računarskoj grafici ne omogućuju trodimenzionalne oblike ili trodimenzionalne optičke fenomene kao što su osvjetljenje, sjene, refleksije i refrakcije svijetlosti itd.. Međutim, pružaju jednu dosta zgodnu mogućnost a to je podjela grafičkih elemenata u slojeve (engl. *layers*) koji se mogu nezavisno prikazivati i editovati za kreiranje efektivnijih i složenijih 2D modela. O slojevima možemo razmišljati kao o providnim folijama koje poredane u specifičnom redoslijedu jedna na drugu čine jedinstvenu sliku. Redoslijed slojeva obično je predstavljen brojem (odnosno „dubinom" sloja, tj. njegovom udaljenošću od posmatrača) i od njega umnogome zavisi krajnji rezultat odnosno prikaz.

Tehnika slojeva odnosno mogućnost neovisnog editovanja i manipulisanja različitih elemenata slike često se naziva i 2.5D grafika i veoma je zastupljena u današnjim aplikacijama za obradu i manipulaciju računarske grafike.

1.4.1.5. PROGRAMI ZA RAD SA RASTERSKIM SLIKAMA

Postoji jako puno programa (na stotine), koji se mogu koristiti za pravljenje ili obradu rasterskih slika. Od najpoznatijih tu su *Adobe Photoshop* i *Corel Photo-Paint* a od besplatnih tu je uvijek prisutni *MS Paint*, potom *GIMP* itd.. Veći editora rasterskih slika koriste RGB model boja dok neki dozvoljavaju upotrebu i drugih modela boja poput CMYK.

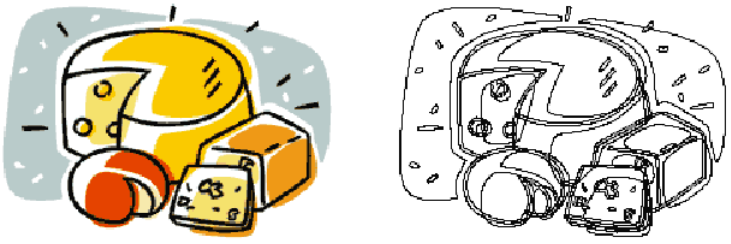
1.4.1.6. FORMATI RASTERSKIH DATOTEKA

Rasterska informacija se može čuvati u velikom broju formata. Evo nekoliko njih:

- BMP: osnovni format rasterske slike bez kompresije
- EPS: fleksibilan format, koji može da sadrži i rasterske i vektorske podatke.
- GIF: često se koristi za Web grafiku, umjesto njega danas se sve više koristi PNG
- JPEG: najpopularniji format opšte namjene. Glavna prednost mu je izuzetno dobra kompresija kolor fotografija.
- PDF: univerzalni format, može da sadrži podatke bilo kog tipa, u pojedinim područjima postepeno potiskuje EPS.
- PICT: format koji može da sadrži i rasterske i vektorske podatke, koristi se na Macintosh računarima.
- TIFF: najpopularniji rasterski format u pripremi za štampu (engl. *desktop publishing* – DTP).

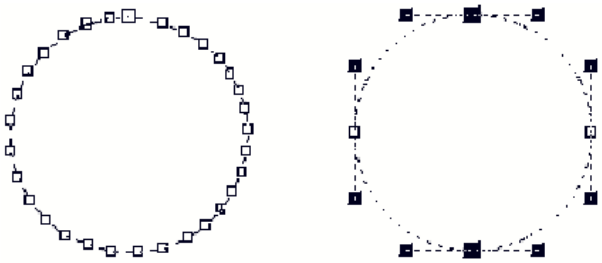
1.5. VEKTORSKA GRAFIKA

Vektorske slike se potpuno opisuju pomoću matematičkih formula. Na Slici 10. lijevo vidi se sama grafika, a desno su linije, koje grade sliku.



Slika 10. Primjer vektorske grafike

Svaka linija se sastoji ili od velikog broja tačaka i linija, koje ih povezuju, ili od manjeg broja kontrolnih tačaka povezanih Bézier-ovim krivama. Drugi metod daje najbolje rezultate i koristi se u većini programa za vektorsko crtanje.



Slika 11. Primjeri kreiranja vektorske grafike

Slika 11. demonstrira oba metoda. Lijevi krug je sastavljen od velikog broja tačaka, koje su spojene pravim linijama. Desni krug je nacrtan pomoću četiri kontrolne tačke (čvora).

Drugi primjer, razmotrimo krug nekog radijusa r. Glavni podaci koje računarski program treba da zna kako bi nacrtao krug su:

1. radijus r
2. koordinatnu poziciju centralne tačke kruga
3. stil i boju linije (može biti i providna)
4. stil i boju punjenja objekta (može biti i providno)

Prednosti ovakvog načina crtanja nad rasterskom grafikom su:

- Ovako mala količina informacija omogućuje mnogo manju veličinu datoteke
- Mogućnost približavanja (*zoom*) bez gubitka na kvaliteti
- Sve ove informacije su zapamćene i mogu se kasnije mijenjati, to znači da pomjeranje, skaliranje, rotiranje i popunjavanje, itd., ne smanjuju kvalitet crteža kao kod rasterske slike.

1.5.1. KARAKTERISTIKE VEKTORSKIH SLIKA

Vektorska grafika je savršena za jednostavne ili složene crteže koji ne treba da budu foto-realistične. Za obradu vektorske slike najčešće se koriste slijedeći programi: *CorelDRAW, Adobe Illustrator* ili *Inkscape*.

Vektorske slike obično zauzimaju malo prostora, zato što sadrže samo Bézier-ove krive, koje stvaraju sliku. To omogućuje prepoznavanje značenja objekata, odnosno daje velike mogućnosti preoblikovanja i promjene svojstava objekata. Vektorsku grafiku je, u principu, moguće uveličavati i umanjivati bez gubitka kvaliteta. To je čini idealnom za logotipe preduzeća, geografske karte, i druge objekte, kojima je često potrebno mijenjati veličinu.

Ipak, postoje izvjesna ograničenja:

- Kod prevelikog smanjivanja mogu nestati tanke linije. Tačnije, one će i dalje postojati ali se ne mogu odštampati ili prikazati na ekranu.
- Male greške mogu postati primjetne kad se slika puno uveliča.
- Mnogi programi omogućavaju da se u vektorskoj grafici koriste i rasterski podaci. Za njih važe ista pravila kao i za sve ostale rasterske slike.

Prije ili kasnije, sva vektorska grafika mora biti prebačena u rastersku kako bi bila prikazana na digitalnom monitoru. Osim toga, vektorske slike nije lako štampati. Posebno su problematični mozaici (male slike koje se ponavljaju u datoteci stotinama ili čak hiljadama puta) i razni grafički efekti.

To ponekad dovodi do suviše komplikovane datoteke, pa se umjesto standardnih štampača koriste tzv. ploteri – štampači posebno namijenjeni za štampanje vektorske grafike.

1.5.2. PROGRAMI ZA RAD SA VEKTORSKOM GRAFIKOM

I ovih programa ima puno, ali su ipak znatno rjeđi nego oni za rasterske slike. Za obradu vektorske slike najčešće se koriste slijedeći programi: *CorelDRAW, Adobe Illustrator, Inkscape* itd..

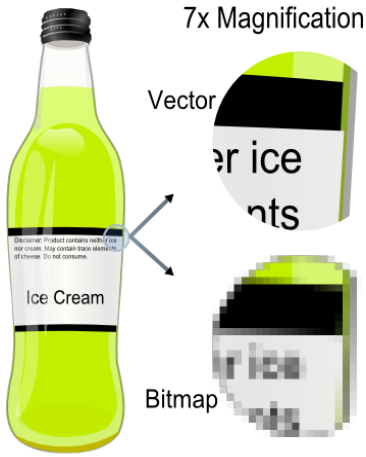
1.5.3. FORMATI VEKTORSKIH DATOTEKA

Vektorskih formata ima nešto manje. Evo nekoliko njih:

- EPS: najpopularniji vektorski format u stolnom izdavaštvu, koji može da sadrži i rasterske i vektorske podatke
- PDF: sve češći univerzalni format
- PICT: i dalje popularan na *Macintosh*-u.
- AI: format *Adobe Illustrator*-a, interno vrlo sličan EPS-u
- CDR: format programa *Corel Draw*
- SVG: nov vektorski format namijenjen za Web

1.5.4. RAZLIKA IZMEĐU VEKTORSKE I RASTERSKE GRAFIKE

Iz ranije navedenih karakteristika vektorske i rasterske grafike moguće je uočiti njihove brojne razlike. Vektorska grafika koja elemente slike opisuje geometrijskim oblicima i matematičkim formulama je u potpunosti suprotna od rasterske grafike koja sliku predstavlja pomoću matrice piksela.



Slika 12. Primjer uvećanja dijela slike kod vektorske i rasterske grafike

Najočiglednije razlike između navedenih sistema jesu te da je za razliku od rasterskih, vektorsku grafiku moguće povećavati i smanjivati bez gubitka kvalitete (Slika 12.) kao i to da vektorska grafika s obzirom na način pohranjivanja vizuelnih informacija zauzima manje memorijskog prostora nego što je to slučaj kod rasterske grafike.

1.6. REZOLUCIJA DIGITALNE SLIKE

Termin rezolucija slike opisuje detalje koja neka slika sadrži i odnosi se na digitalne, filmske i mnoge druge vrste slika. Rezolucija slike je važna zato što određuje njen kvalitet – veća rezolucija znači i više detalja na slici.

Rezolucija slike može se izmjeriti na različite načine. U osnovi, rezolucija definiše koliko međusobno blizu tačke (linije u filmskoj-televizijskoj slici) mogu biti a i dalje budu vidljive. U zavisnosti gdje se slika koristi, jedinica mjere njene rezolucije mogu biti data kao fizička veličina (tačaka po inču/centimetru) ili kao broj tačaka/linija cjelokupne slike.

Termin rezolucija kod digitalne slike se odnosi na broj piksela (tačaka) digitalne slike. Konvencijom je usvojeno da se za sliku od M piksela širine i N piksela visine kaže da ima rezoluciju MxN. Drugi način jeste da se rezolucija izrazi preko ukupnog broja piksela, obično kao broj megapiksela, koji se dobije kada se ukupan broj piksela slike po visini pomnoži sa ukupnim brojem piksela po širini i podijeli sa milion. Na primjer, slika 2048 piksela širine i 1536 piksela visine ima rezoluciju 2048 * 1536 = 3 145 728 piksela ili 3.1 megapiksela.

Još jedan popularni način izražavanja rezolucije digitalne slike jeste preko DPI/PPI (*engl*. Dots Per Inch/Pixel Per Inch) faktora. DPI/PPI označava stvarnu gustinu tačaka slike kada se ona reproducira na nekom fizičkom uređaju, npr. odštampa na papiru ili prikaže na monitoru, te ustvari predstavlja rezoluciju reprodukovane a ne stvarne slike. Promjenom DPI faktora za sliku poznate rezolucije moguće je promijeniti veličinu njenog ispisa na papiru dok rezolucija originalne digitalne slike ostaje ista. Digitalno pohranjena slika nema svoju fizičku dimenziju, mjerenu u inčima ili centimetrima tako da ni DPI ni PPI veličina ne mijenja kvalitet digitalne slike. Rezolucija digitalne slike jeste i ostaje njena dimenzija u pikselima.

Za vektorske slike ne postoji PPI faktor pošto su one neovisne od rezolucije (svi ispisi imaju isti kvalitet). Međutim, postoji nešto što se zove željena veličina (engl. *target printing size*). Neki formati slika, poput Photoshop-ovog PSD formata, može sadržavati rasterske i vektorske podatke unutar istog dokumenta. U tim slučajevima, promjena PPI faktora znači promjenu na željenu veličinu rasterskog dijela slike koji opet dovodi do promjene vektorskog dijela slike kako bi se zadržao međusobni odnos veličina između rasterskih i vektorskih elemenata unutar slike.

Drugi formati, poput PDF-a, su primarno vektorski formati koji mogu sadržavati bitmape. Kod njih promjena u željenoj veličini ispisa mijenja PPI faktor rasterskog dijela slike. To je obrnuti princip rada nego kod rasterskih formata poput *Photoshopa* ali koji daje isti rezultat – održavanje konstantnih proporcija između rasterskih i vektorskih elemenata slike.

1.7. TRODIMENZIONALNA RAČUNARSKA GRAFIKA

3D računarska grafika (za razliku od 2D računarske grafike) koristi trodimenzionalni prikaz geometrijskih podataka (obično u Kartezijevom koordinatnom sistemu) koji se pohranjuju u računaru za potrebe obavljanja proračuna i iscrtavanje 2D slika. Takve slike mogu biti kasnije pregledavane ili posmatrane u stvarnom vremenu.

Unatoč razlikama, 3D računarska grafika oslanja se na mnoge algoritme koji se koriste kod 2D vektorske i rasterske grafike. Sa stanovišta softvera, razlika između 2D i 3D grafike često je nejasna; 2D aplikacije mogu koristiti 3D tehnike kako bi postigle efekte poput osvjetljenja, 3D aplikacije obično koristite 2D *rendering* tehnike prilikom iscrtavanja objekata.

Proces kreiranja 3D računarske grafike se može redoslijedom podijeliti na 3 osnovne faze:

- 3D modeliranje koje opisuje proces formiranja oblika objekta
- računarska animacija koja opisuje kretanje i položaj objekata unutar scene
- 3D renderiranje koje stvara sliku objekta

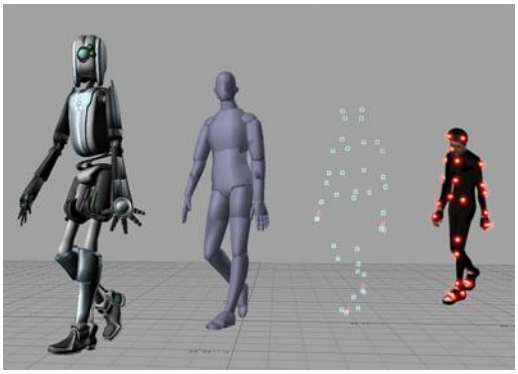
1.7.1. 3D MODELIRANJE

predstavlja proces razvoja matematičke reprezentacije bilo kojeg trodimenzionalnog objekta putem specijaliziranog softvera. Krajnji rezultat 3D modeliranja naziva se **3D model**. Može biti prikazan kao dvodimenzionalna slika kroz proces 3D iscrtavanja (engl. *rendering*) ili se može koristiti u računarskoj simulaciji. Model može biti i fizički stvoren koristeći se specijaliziranim uređajima za 3D printanje. Modeli mogu biti kreirani automatski ili ručno. Ručni proces modeliranja koji priprema geometrijske podatke za 3D računarsku grafiku je sličan plastičnom umijeću kao što je kiparstvo.

Slika 13. 3D model izrađen i iscrtan upotrebom *Blender*-a.

1.7.2. RAČUNARSKA ANIMACIJA

Računarska animacija je umijeće kreiranja pokretnih slika upotrebom računara.



Slika 14. Računarska animacija kreirana *motion capture* tehnikom

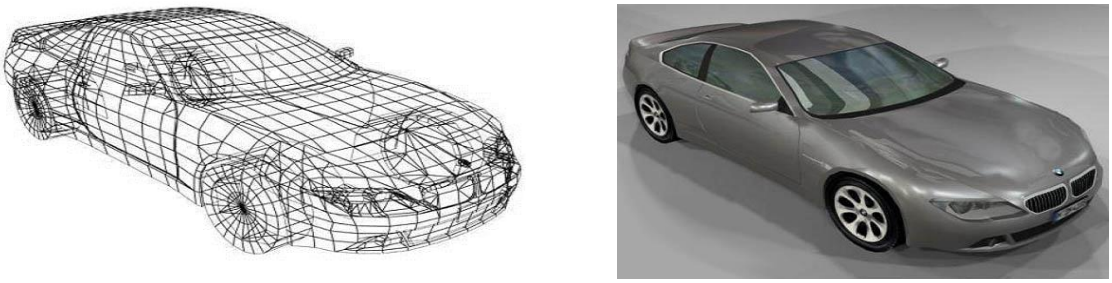
To je potkategorija računarske grafike. Ponekad je rezultat animacije prikaz na samom računaru, ali ponekad je kao rezultat potreban i drugi medij, kao npr. film. Za računarski animaciju se često koristi termin CGI (engl. *computer-generated imaging*) posebno kada se koristi u filmu. Kako bi se stvorila iluzija pokreta, slika je prikazana na zaslonu računara i onda brzo zamijenjena novom slikom koja je slična prethodnoj, ali neznatno pomaknuta. Ova tehnika je identična iluziji pokreta na televiziji i pokretnim slikama.

1.7.3. RENDERIRANJE

Renderiranje (engl. *rendering*) ili iscrtavanje je proces kreiranja konačne 2D slike ili animacije iz pripremljene scene, odnosno 3D objekta. To znači da prije nego što se objekti iscrtaju, oni moraju biti postavljeni (engl. *layout*) unutar scene, tj. potrebno je odrediti prostorni odnos između objekata u sceni, uključujući lokaciju i veličinu.

Renderiranje možemo porediti sa slikanjem ili snimanjem video klipa nakon što završimo sa pripremom scene. Razvijeno je nekoliko različitih metoda za renderiranje, krenuvši od foto-nerealističnih kao što je žičani okvir (engl. *wireframe*), preko renderiranja baziranog na poligonima pa do kompleksnijih algoritama kao što su trasiranje zrake (engl. *ray tracing*) i bacanje zrake (engl. *ray casting*).

Renderiranje može trajati od djelića sekunde, pa do nekoliko dana za jednu sliku ili okvir. Na Slici 15. prikazan je efekat iscrtavanja 3D objekta.



a) 3D *wireframe* model **b) 3D model baziran na poligonima**

Slika 15. Različiti načini iscrtavanja 3D modela

1.8. BOJE I TONOVI

Nosilac vizuelne informacije je svjetlost. Da bismo objasnili pojave i veličine vezane za svjetlost napraviti ćemo podjelu na neobojenu ili bijelu svjetlost i obojenu svjetlost.

Kada kažemo "crno-bijela" slika ne mislimo uvijek na sliku koja je sastavljena isključivo od crnih i bijelih površina. U engleskom informatičkom rječniku postoje zato dva izraza za ovakve slike: "Black&White" za slike čiji su elementi zaista ili crni ili bijeli i "Grayscaled" za slike sastavljene od elemenata crne, bijele ili raznih nivoa (tonova) sive boje. Jedina bitna karakteristika svakog elementa ovakvih slika je količina bijele svjetlosti koju emituje.

Količina svjetlosti može se opisati na više načina, npr. kao **INTENZITET** svjetlosti što je fizikalna veličina i odgovara energiji koju nosi ili kao **OSVJETLJENOST** (engl. *Brightness*) što je psihološka odrednica, stvar ljudske percepcije, te kao takva, mnogo značajnija za računarsku grafiku od intenziteta svjetlosti.

Tako je za opis crnobijele slike dovoljno dati osvijetljenost svakog njenog elementa. Postavlja se pitanje šta je crno a šta bijelo na jednoj takvoj slici, a odgovor u računarskoj grafici nije jednostavan. U idealnom slučaju, crno je potpuni nedostatak svjetlosti a bijelo je maksimalna osvijetljenost neobojenom svjetlošću. U praksi nema idealnih slučajeva pa se za crnu boju uzima minimalno osvjetljenje, a za bijelu boju maksimalno osvjetljenje neobojenom svjetlošću koje grafički uređaj može dati. Odnos između ove dvije vrijednosti naziva se **DINAMIČKI RASPON** (engl. *dynamic range*) i bitna je karakteristika svih grafičkih uređaja (kao i svih drugih uređaja i medija koji se koriste za prezentaciju i vizuelizaciju).

Najjednostavnije je skalu sivih tonova formirati tako da se ovi poredaju linearno između bijele i crne boje. Ovakav raspored, međutim, ne daje zadovoljavajuće rezultate, zbog nelinearne prirode oka. Ljudsko oko bolje uočava razlike između tamnih nego između svijetlih tonova pa zato sivi tonovi u dijelu skale bliže crnoj boji moraju biti gušće poredani nego u svijetlom dijelu. Jedna od raspodjela sivih tonova koja ovo zadovoljava je logaritamska (sljedeći položaj na skali se dobija množenjem prethodnog sa konstantom, a ne dodavanjem konstante).

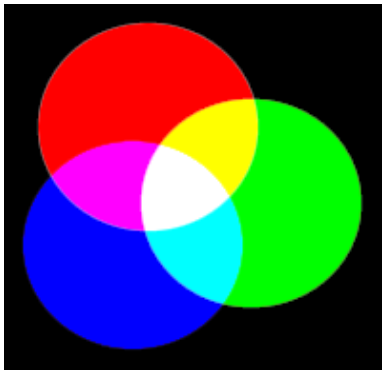
Osobina svjetlosti nije samo njen intenzitet nego i boja. Fizikalno, svjetlost je talas i kada u tom smislu govorimo o boji svjetlosti, govorimo u stvari o frekvenciji tog talasa. Sa druge strane, kada u računarskoj grafici govorimo o boji, govorimo o čovjekovom subjektivnom doživljaju svjetlosti, bez obzira na njenu prirodu.

1.8.1. MODELI BOJA

Bijela svjetlost je složena svjetlost. Može se razbiti na spektar obojenih svjetlosnih komponenti ili dobiti slaganjem odgovarajućih obojenih svjetlosti. Ako se smanji ili poveća intenzitet neke od komponenti bijele svjetlosti dobija se složena obojena svjetlost. Drugim riječima, kombinujući više obojenih svjetlosti može se dobiti svjetlost čija je boja drugačija od boje svake komponente te svjetlosti. Mijenjajući intenzitet ovih komponenti mijenjamo boju složene svjetlosti. U ovom slučaju, rezultujuća boja dobijena je sumiranjem boja svih komponenti svjetlosti pa govorimo o **ADITIVNOM** miješanju boja.

1.8.2. RGB MODEL

Model RGB, koji koriste računarski monitori i TV ekrani, dodjeljuje vrijednosti od 0 do 255 svakoj od tri RGB komponente (crvenoj, zelenoj i plavoj boji). Kada su vrijednosti sve tri komponente 255, dobija se bijela boja, a kada su te vrijednosti 0 - crna. U ovom kontekstu, vrijednost označava relativnu jačinu boje.



Slika 16. Aditivno miješanje boja - RGB kolor model

Sistem aditivnog miješanja boja u kojem su **Crvena** (*Red*), **Zelena** (*Green*) i **Plava** (*Blue*) osnovne boje naziva se "RGB sistem" ili "RGB model". Svaka boja u RGB sistemu dobija se miješanjem, u određenim odnosima, crvene, zelene i plave boje. U RGB modelu, zastupljenost svake osnovne boje može se opisati brojem, npr. procentom u odnosu na maksimalnu zastupljenost (100%), te se tako svaka boja može opisati pomoću tri broja. Obično se za predstavljanje svake od boja koristi po jedan bajt informacija pa se opseg vrijednost koji se dodjeljuje nekoj od osnovnih boja kreće od 0 do 255. Kada su vrijednosti sve tri komponente 255, dobija se bijela boja, a kada su te vrijednosti 0 - crna.

Ako su sve tri osnovne boje RGB modela podjednako zastupljene, rezultujuća boja je neka nijansa sive, zavisno od intenziteta osnovnih boja. Ovaj model boja koriste računarski monitori i TV ekrani za prikaz slike u boji.

1.8.3. CYMK MODEL

Ukoliko posmatramo drugi princip kreiranja boja kod kojeg se razne boje dobijaju tako što od bijele boje „oduzmemo" dio sprektra koji predstavlja neka druga boja dobija se miješanje boja koje se naziva **SUBSTRAKTIVNO**. Za osnovne boje u substraktivnom modelu miješanja boja obično se uzimaju boje komplementarne osnovnim bojama u RGB modelu: **Cijan** (*Cyan*), **Purpurnocrvena** (*Magenta*) i **Žuta** (*Yelow*), pa se ovaj model često naziva i "CMY model".

Slika 17. Substraktivno miješanje boja - CMY kolor model

Miješanjem osnovnih boja CMY modela u jednakom omjeru dobijaju se razne nijanse sive, zavisno od intenziteta osnovnih boja, ili crna, ako se upotrijebe maksimalni intenziteti osnovnih boja.

U teoriji ovo je potpuno tačno, međutim, praktično je gotovo nemoguće dobiti potpuno crnu boju miješajući tri osnovne boje CMY modela. Zato se često uz cijan, purpurnocrvenu i žutu koristi još i crna komponenta, i tada govorimo o CMYK kolor modelu.

Upotreba RGB kolor modela prirodna je kod uređaja i medija koji emituju svjetlost (displeji, projektori i sl.), dok se CMY ili CMYK kolor modeli koriste kod uređaja i medija koji nisu aktivni izvori svjetlosti (npr. slike otisnute na papiru pomoću kolor štampača, plotera i sl.).

1.8.4. HSV MODEL

Oba gore predstavljena kolor modela orjentisana su prema prirodi svjetlosti, tj. uvažavaju prirodni, fizikalni način miješanja boja ali ne i prirodu čovjekovog uočavanja i razlikovanja boja. Ono što čovjek uočava posmatrajući obojene predmete ili površi su sama boja tj. njen ton (engl. *Hue*), zasićenje te boje (engl. *Saturation*) i osvijetljenost (engl. *Brightness*), a ne intenzitet osnovnih boja. Na osnovu ovoga napravljen je kolor model koji je čovjeku puno jednostavniji i pogodniji za upotrebu, tzv. HSB kolor model. Ovaj kolor model se često naziva i HSV ili HSL ili HSI kolor model, razlika je samo terminološka, jer se za osvjetljenost umjesto izraza *Brightness* koriste izrazi *Value*, *Lightness* i *Intensity* respektivno.

Pojednostavljenje u korištenju HSB kolor modela može se ilustrovati jednostavnim primjerom: ako bi, na primjer, željeli promijeniti samo osvjetljenost nekog elementa slike, bez promjene boje, u RGB ili CMY kolor modelu morali bi podešavati intenzitete svih komponenti boje i to nelinearno, dok se u HSB modelu to postiže jednostavnom promjenom samo jednog parametra boje (*Brightness*).

1.8.5. PRIMJENA RAZLIČITIH MODELA BOJA

Zbog prirode tehnologija korištenih u raznim grafičkim uređajima, danas se za akviziciju ili prikaz slika u većini slučajeva koristi RGB ili CMY (CMYK) kolor model, zavisno od uređaja. Tako, svi skeneri digitalizuju sliku koristeći RGB kolor model, sve komercijalne kolor displej tehnologije su bazirane na RGB modelu, kolor štampači i ploteri uglavnom koriste CMY ili CMYK kolor model itd..

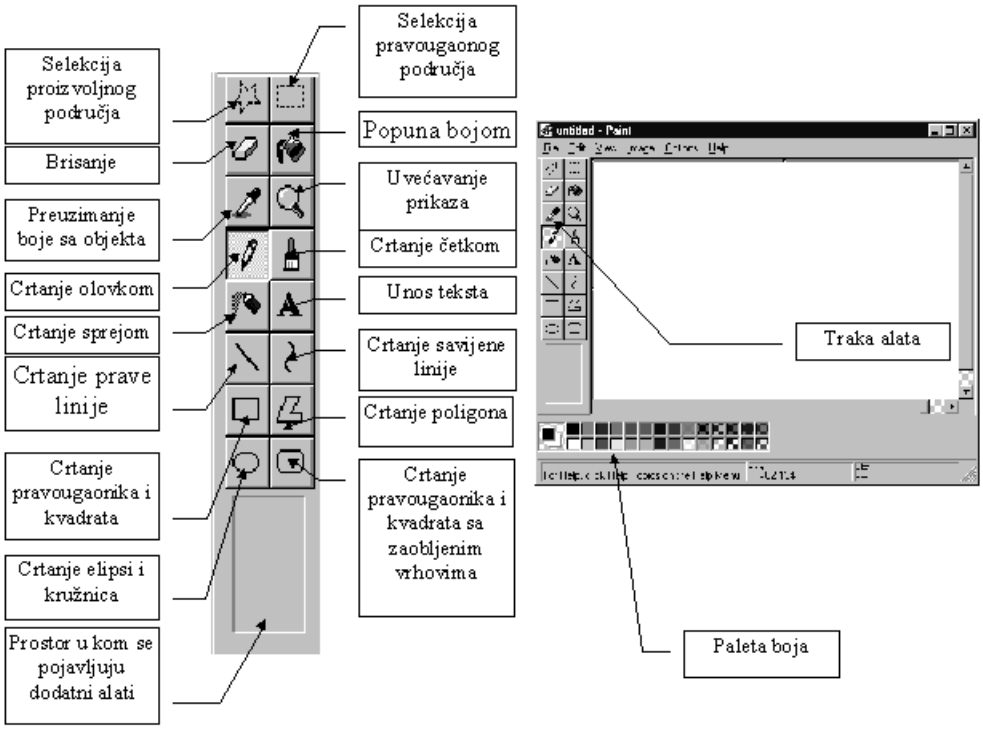
HSB (HSV) kolor model se zbog jednostavnosti upotrebe uglavnom koristi interno, u programima za kreiranje ili obradu slika, ali ga kao osnovu koriste i neki formati za digitalni zapis slika (JPEG).

Odavde se vidi da se često ukazuje potreba za konverzijom slika iz jednog kolor modela u drugi. Ova konverzija nije trivijalna. Zbog specifičnosti svakog od ovih kolor modela i uređaja koji te modele koriste, kao i zbog ograničene veličine podataka kojima se u računarskoj grafici opisuje boja, nije moguće izvršiti direktno, linearno preslikavanje jednog kolor modela u drugi i pri tome dobiti zadovoljavajuće rezultate. Ovo preslikavanje se vrši složenim algoritmima ili, češće, pomoću krivulja preslikavanja koje se mogu prilagođavati zahtjevima i mogućnostima raznih ulaznih i izlaznih grafičkih uređaja.

Nabrojani kolor modeli se najčešće sreću, ali nisu i jedini. Mnoge grane grafičke djelatnosti koriste specifične kolor modele čija je upotreba vezana za uzak krug aplikacija i korisnika, pa ovdje neće biti detaljnije objašnjavane.

1.9. PROGRAM MS PAINT

Program *MS Paint* je namijenjen za kreiranje grafičkih fajlova u *bitmap* formatu (BMP). Pokreće se iz sistema podmenija startnog menija Windows okruženja: Meni *Start - Programs - Accessories - Paint.* Takođe se može pokrenuti i na druge načine objašnjenje u dijelu koji obrađuje pokretanje programa.



Slika 18. Radno okruženje MS PAINT-a

CRTANJE LINIJE

1. Klik na odgovarajuću tipku u okviru sa alatima (u okviru ispod okvira sa alatima kliknuti na ponuđene debljine linije, kliknuti na boju na paleti boja za boju linije)
2. Dovesti pokazivač miša na poziciju gdje će linija počinjati i pritisnuti tipku miša, a zatim ne puštajući tipku povući liniju do željene pozicije

Da bi se nacrtala pravilna horizontalna, vertikalna ili linija pod uglom od 45 stepeni, treba držati pritisnutu tipku SHIFT dok se linija povlači.

CRTANJE OLOVKOM

1. Klik na odgovarajući alat pa klik na željenu boju
2. Držeći pritisnutu tipku miša povlačiti liniju

CRTANJE SAVIJENE LINIJE

1. Klik na odgovarajući alat, klik na debljinu linije, klik na boju
2. Crtanje prave linije povlačenjem miša, klik pored linije da joj se proizvede zakrivljenost koja zavisi od mjesta gdje se kliknulo (maksimalno dva puta)

CRTANJE ELIPSE I KRUŽNICE

1. Klik na odgovarajući alat
2. Klik na boju za linije
3. Klik desnom tipkom na boju za prostor elipse ili kruga, klik na stil popunjavanja prostora (ispod okvira alata)
4. Za crtanje elipse povlačiti pokazivač miša dijagonalno, a za crtanje kružnice/kruga pri tome držati pritisnutu tipku SHIFT.

CRTANJE PRAVOUGAONIKA I KVADRATA

1. Klik na odgovarajući alat (prazni ili popunjeni oblik - zaobljenih ili oštrih uglova)
2. Klik na boju za liniju
3. Klik desnom tipkom miša na boju za popunu prostora objekta
4. Za crtanje pravougaonika povlačiti pokazivač miša u željenom pravcu dijagonalno, a za kvadrate držati pritisnutu tipku SHIFT pri toj radnji.

CRTANJE POLIGONA

1. Klik na odgovarajući alat, klik na boju za liniju
2. Za crtanje popunjenog poligona (obojene pozadine) kliknuti desnom tipkom miša na željenu boju, pa kliknuti na stil popune (ispod okvira alata)
3. Crtati poligon tako što se klikne na svaki budući ugao poligona, dvostruki klik završava poligon. Držeći pritisnutu tipku SHIFT postiže se crtanje poligona sa uglovima od 45 i 90 stepeni.

UBACIVANJE I FORMATIRANJE TEKSTA

1. Klik na odgovarajući alat
2. Kreirati okvir u kojem će se ispisivati tekst povlačenjem pokazivača miša dijagonalno, odabrati font, veličinu i stil koji se želi u otvorenom okviru,
3. Kliknuti unutar definisanog okvira i otkucati tekst. Veličina okvira se može mijenjati a okvir premjestiti na drugu poziciju. Klik na boju u paleti boja definiše boju slova. Da bi se promijenila boja pozadine na kojoj se slova ispisuju, kliknuti desnom tipkom miša na boju u paleti boja.
4. Po završetku kucanja kliknuti izvan okvira za tekst.

POPUNA BOJOM

1. Klik na odgovarajući alat
2. Klik na područje koje treba obojiti

CRTANJE ČETKOM

1. Klik na odgovarajući alat, klik na oblik traga četke ispod okvira alata,
2. Klik na boju iz palete boja,
3. Bojenje se vrši povlačenjem pokazivača miša

CRTANJE SPREJOM

1. Klik na odgovarajući alat, klik na veličinu spreja ispod okvira alata, klik na željenu boju
2. Bojenje se vrši povlačenjem pokazivača miša.

PREUZIMANJE BOJE DRUGOG OBJEKTA

1. Klik na odgovarajući alat
2. Klik na objekat koji je obojen željenom bojom
3. Klik na objekat koji se namjerava obojiti

SELEKCIJA DIJELA SLIKE

1. Za selekciju pravougaone površine: klik na odgovarajući alat i povlačenje miša dijagonalno,
2. Za selekciju nepravilne površine: klik na odgovarajući alat i povlačenje miša okolo željene površine

BRISANJE MANJIH POVRŠINA

1. Klik na odgovarajući alat, klik na veličinu brisača ispod okvira alata
2. Brisanje povlačenjem pokazivača miša.

BRISANJE VEĆIH POVRŠINA

1. Klik na jedan od alata za selekciju i selekcija površine povlačenjem pokazivača miša,
2. Kliko desnom tipkom miša na boju izavaće promjenu boje označenog dijela slike,
3. U meniju *Edit* klik na opciju *Clear Selection* da se područje deselektuje.

KOPIRANJE DIJELA SLIKE

1. Selektovati željeni dio, kliknuti na jednu od ponuđenih mogućnosti (ispod okvira alata): selekcija sa bojom pozadine ili bez nje (sa transparentnom pozadinom)
2. Klik na *Copy* u meniju *Edit*
3. Klik na *Paste* u meniju *Edit*
4. Premjestiti dobijenu kopiju selektovanog dijela slike povlačenjem na željenu poziciju
5. Da se dio deselektuje kliknuti izvan selektovanog dijela.

PROMJENA VELIČINE SLIKE

1. U meniju *Image* klik na opciju *Attributes*,
2. Klik na mjernu jedinicu u dobijenom okviru
3. Kucanje vrijednosti za dužinu i širinu

UVEĆAVANJE I UMANJIVANJE PRIKAZA SLIKE

1. U meniju *View* izabrati opciju *Zoom*, a zatim u dobijenom meniju kliknuti na *Normal Size* (normalan prikaz), *Large Size* (uvećan prikaz) ili *Custom* (definisanje veličine prikaza).

OKRETANJE I ROTIRANJE SLIKE

1. Selektovati željeni dio slike
2. U meniju *Image* kliknuti na *Flip/Rotate*
3. Kliknuti na željenu opciju u dobijenom okviru

RAZVLAČENJE I NAGINJANJE SLIKE

1. Selektovati željeni dio slike
2. U meniju *Image* kliknuti na *Stretch/Skew*
3. Kliknuti na željenu opciju u dobijenom okviru

PRENOS SLIKA NA DESKTOP

1. Da se slikom pokrije kompletan desktop tako što se ona ponavlja na desktopu koliko je potrebno za to, u meniju *File* kliknuti na opciju *Set As Wallpaper (Tiled)*,
2. Da se slika centrira na desktopu, u meniju *File* kliknuti na opciju *Set As Wallpaper (Centered)*. Slika mora prethodno biti snimljena.

2. BAZE PODATAKA

2.1. UVOD

2.1.1. POJAM BAZE PODATAKA

Baza podataka je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računara. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima. Unos, izmjena, brisanje i čitanje podataka obavlja se posredstvom zajedničkog softvera. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičke organizacije podataka, već se upućuju na logičku organizaciju podataka. Ulaze u sve širu primjenu tokom 70-tih godina XX vijeka.

Sistem za upravljanje bazom podataka (engl. *Data Base Management System - DBMS*) je poslužitelj (server) baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Također, on obavlja u ime klijenata sve operacije s podacima. Dalje, on je u stanju podržati razne baze, od kojih svaka može imati svoju logičku strukturu, no u skladu s istim modelom. Isto tako, brine se za sigurnost podataka, te automatizira administrativne poslove s bazom. Microsoft Access je jedan od primjera sistema za upravljanje bazama podataka. Pored ovog, postoji i niz drugih, kao što su: Microsoft SQL, MySQL, Oracle itd.. Osnovne funkcije sistema za upravljanje bazama podataka su:

- fizička organizacija i pohranjivanje podataka
- kreiranje baze podataka
- pristup informacijama u bazi podataka: unos, brisanje, izmjena, pregled
- zaštita podataka

Podaci u bazi su logički organizovani u skladu s nekim **modelom podataka**. Model podataka je skup pravila koja određuju kako može izgledati logička struktura baze. Model čini osnovu za koncipiranje, projektovanje i implementiranje baze.

Od 80-tih godina pa sve do današnjih dana preovladava relacioni model. Očekivani prelaz na objektni model za sada se nije desio, tako da današnje baze podataka uglavnom još uvijek možemo poistovjetiti s relacionim bazama.

Baze podataka su zapravo tabele u kojima držimo podatke. Bazu podataka možete koristiti za kućnu upotrebu, vođenje nekih računovodstvenih poslova, ispisivanje podataka i izvještaja, ali češće se koristi za spremanje i čitanje podataka koje koriste programi. U takvim situacijama više programa može koristiti podatke jedne baze podataka. MS Access je jedan od programa za izradu baza podataka.

Baze podataka su osnovna forma memorisanja podataka današnjice, kako u "**online**", tako i u "**offline**" svijetu. Koriste se za memorisanje miliona različitih **kombinacija/tipova informacije** kao što su podaci o proizvodima, zaposlenim, personalne, adresne informacije, itd..

Prije nego što se počne kreirati baza podataka, nužno je razumjeti osnovni koncept i teoretske osnove: zašto se baze podataka koriste i kako se kreiraju. Na ovim stranicama biti će pojašnjeno šta je baza podataka, metodologija kreiranja baze podataka, modeliranje podataka i osnove korištenje Microsoft Access-a.

Koncepcija baze podataka polazi od stanovišta da treba stvarati jedinstveni skup podataka, time da između podataka postoje određeni odnosi. Jedan te isti skup podataka, prema tome služi većem broju aplikacija, odnosno korisnika.

Baze podataka se stoga mogu definisati i kao skup međusobno povezanih, međusobno usklađenih podataka, bez nepotrebne redundancije. Ili šire i preciznije rečeno: Baza podataka je "*…skup međusobno povezanih podataka, skladištenih zajedno, bez štetne ili nepotrebne redundancije, da bi na optimalan način služila jednoj ili većem broju aplikacija odnosno korisnika*".

Podaci se memorišu na takav način, da budu neovisni od programa koji se njima služe. Za dodavanje novih podataka, odnosno modifikaciju i pretraživanje postojećih, koristi se opšti kontrolisan pristup. To znači da ta organizacija omogućuje optimalnu upotrebu podataka, koji se samo jednom zapisuju i s jednog mjesta ažuriraju, pa mogu da služe većem broju korisnika i aplikaciju.

U osnovi, bazu podataka možemo definisati kao organizovanu kolekciju podataka. Baza podataka čuva i organizuje informacije. Informacija je osnova poslovanja bilo koje kompanije ili organizacije, te je efikasna obrada informacija nužna za uspjeh njihovog poslovanja.

Baze podataka koriste upite ("*query*"). Upiti omogućavaju pregled podataka, unos i izmjenu podataka, i druge poslove vezane za korištenje podataka u bazi. Upiti se grade korištenjem posebnog programskog jezika SQL (engl. ***S**tructured **Q**uery **L**anguage*). Ovaj programski jezik također omogućava i kreiranje osnovne strukture baza podataka.

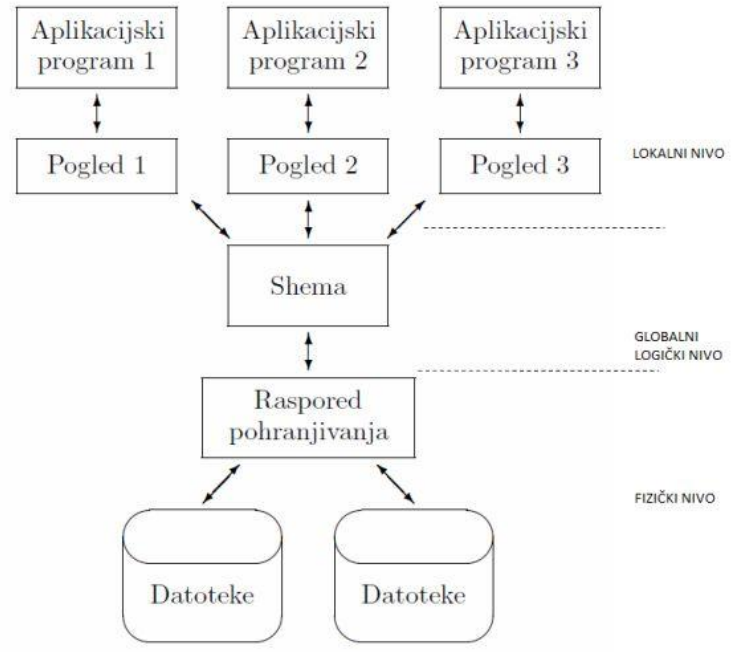
2.2. CILJEVI KOJI SE POSTAVLJAJU PRED BAZU PODATAKA

Baze podataka predstavljaju viši nivo rada s podacima u odnosu na klasične programske jezike. Taj viši nivo rada očituje se u tome što tehnologija baza podataka nastoji (i u velikoj mjeri uspijeva) ispuniti sljedeće ciljeve.

- **Fizička nezavisnost podataka**
  Razdvaja se logička definicija baze od njene stvarne fizičke implementacije. Znači, ako se fizička struktura promijeni (na primjer, podaci se prepišu u druge datoteke na drugim diskovima), to neće zahtijevati promjene u postojećim aplikacijama.
- **Logička nezavisnost podataka**
  Razdvaja se globalna logička definicija cijele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se logička definicija promijeni (na primjer uvede se novi zapis ili veza), to neće zahtijevati promjene u postojećim aplikacijama. Lokalna logička definicija obično se svodi na izdvajanje samo nekih elemenata iz globalne definicije, uz neke jednostavne transformacije tih elemenata.
- **Fleksibilnost pristupa podacima**
  U starijim mrežnim i hijerarhijskim bazama, metode pristupa podacima bile su unaprijed definisane, dakle korisnik je mogao pristupati podacima jedino onim redoslijedom koji je bio predviđen u vrijeme projektovanja i implementiranja baze. Danas se zahtijeva da korisnik može slobodno postupati podacima, te po svom nahođenju uspostavljati veze među podacima. Ovom zahtjevu zaista zadovoljavaju jedino relacione baze.
- **Istovremeni pristup podacima**
  Baza mora omogućiti da veći broj korisnika istovremeno koristi iste podatke. Pritom ti korisnici ne smiju ometati jedan drugoga, te svaki od njih treba imati utisak da sam radi s bazom.
- **Čuvanje integriteta**
  Nastoji se automatski sačuvati korektnost i konzistentnost podataka, i to u situaciji kad postoje greške u aplikacijama, te konfliktne (istovremene) aktivnosti korisnika.
- **Mogućnost oporavka nakon kvara**
  Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili grešaka u radu sistemskog softvera.
- **Zaštita od neovlaštenog korištenja**
  Mora postojati mogućnost da se korisnicima ograniče prava korištenja baze, dakle da se svakom korisniku reguliše ovlaštenja tj. precizira šta smije a šta ne raditi s podacima.
- **Zadovoljavajuća brzina pristupa**
  Operacije s podacima moraju se odvijati dovoljno brzo, u skladu s potrebama određene aplikacije. Na brzinu pristupa može se uticati odabirom pogodnih fizičkih struktura podataka, te izborom pogodnih algoritama za pretraživanje.
- **Mogućnost podešavanja i kontrole**
  Velika baza zahtijeva stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj građi, rutinsko pohranjivanje rezervnih kopija podataka, regulisanje prava pristupa korisnika. Također, svrha baze se vremenom mijenja, pa povremeno treba promijeniti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizirano. Odgovorna osoba zove se **administrator** baze podataka (engl. *Database Administrator – DBA*).

2.3. ARHITEKTURA BAZE PODATAKA

Arhitektura baze podataka sastoji se od tri sloja i interfejsa među slojevima, kao što je prikazano na slici



Slika 19. Arhitektura baze podataka

1. **Fizički nivo** odnosi se na fizički prikaz i raspored podataka na jedinicama vanjske memorije. To je aspekt kojeg vide samo sistemski programeri (oni koji su razvili DBMS). Sama fizički nivo može se dalje podijeliti na više podnivoa apstrakcije, od sasvim konkretnih staza i cilindara na disku, do već donekle apstraktnih pojmova datoteke i zapisa kakve susrećemo u klasičnim programskim jezicima.
2. **Raspored pohranjivanja** opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.
3. **Globalni logički nivo** odnosi se na logičku strukturu cijele baze. To je aspekt kojeg vidi projektant baze odnosno njen administrator. Zapis logičke definicije naziva se **šema** (engl. *shema*). Šema baze podataka je tekst ili dijagram koji definiše logičku strukturu baze, i u skladu je sa zadanim modelom. Dakle, imenuju se i definišu svi tipovi podataka i veze među tim tipovima, u skladu s pravilima korištenog modela. Također, šema uvodi i ograničenja kojim se čuva integritet podataka.
4. **Lokalni logički nivo** odnosi se na logičku predodžbu o dijelu baze koji koristi pojedina aplikacija. To je aspekt kojeg vidi korisnik ili aplikacijski programer. Zapis jedne lokalne logičke definicije zove se **pogled** (engl. *view*) ili podšema. To je tekst ili dijagram kojim se imenuju i definišu svi lokalni tipovi podataka i veze medu tim tipovima, opet u skladu s pravilima korištenog modela. Također, pogled zadaje preslikavanje kojim se iz globalnih podataka i veza izvode lokalni.

2.4. JEZICI ZA RAD S BAZAMA PODATAKA

Komunikacija korisnika odnosno aplikacijskog programa i DBMS-a odvija se pomoću posebnih jezika. Ti jezici tradicionalno se dijele na sljedeće kategorije.

- **Jezik za opis podataka** (engl. *Data Description Language - DDL*)
  Služi projektantu baze ili administratoru u svrhu zapisivanja šeme ili pogleda. Dakle tim jezikom definišemo podatke i veze medu podacima, i to na logičkoj razini. Koji puta postoji posebna varijanta jezika za šemu, a posebna za poglede. Naredbe DDL obično podsjećaju na naredbe za definisanje složenih tipova podataka u jezicima poput COBOL, PL/I, C, Pascal.

- **Jezik za manipuliranje podacima** (engl. *Data Manipulation Language - DML*)
  Služi programeru za uspostavljanje veze izmedu aplikacijskog programa i baze. Naredbe DML omogućavaju "manevrisanje" po bazi, te jednostavne operacije kao što su upis, promjena, brisanje ili čitanje zapisa. U nekim softverskim paketima, DML je zapravo biblioteka potprograma: "naredba" u DML svodi se na poziv potprograma. U drugim paketima zaista se radi o posebnom jeziku: programer tada piše program u kojem su izmiješane naredbe dvaju jezika, pa takav program treba prevoditi s dva prevodioca (*DML - precompiler*, obični *compiler*).
- **Jezik za postavljanje upita** (engl. *Query Language - QL*)
  Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specificiraju rezultat kojeg želimo dobiti, a ne i postupak za dobivanje rezultata.

Ovakva podjela na tri jezika danas je već prilično zastarjela. Naime, kod relacionih baza postoji tendencija da se sva tri jezika objedine u jedan sveobuhvatni. Primjer takvog **integrisanog** jezika za relacione baze je SQL (Structured Query Language): on služi za definisanje podataka, manipulisanje i pretraživanje. Integrisani jezik se može koristiti interaktivno (preko on-line interpretera) ili se integriše u aplikacijske programe.

Naglasimo da gore spomenute vrste jezika nisu programski jezici. Dakle ti jezici su nam nužni da bi se povezali s bazom, no oni nam nisu dovoljni za razvoj aplikacija koje će nešto raditi s podacima iz baze.

Tradicionalni način razvoja aplikacija koje rade s bazom je korištenje klasičnih programskih jezika (COBOL, PL/I, C, Pascal . . .) s ugniježdenim DML-naredbama. U 80-tim godinama 20. vijeka bili su dosta popularni i tzv. **jezici 4. generacije** (4-th Generation Languages - 4GL): riječ je o jezicima koji su bili namijenjeni isključivo za rad s bazama, te su zato u tom kontekstu bili produktivniji od klasičnih programskih jezika opšte namjene. Problem s jezicima 4. generacije je bio u njihovoj nestandardnosti: svaki od njih je u pravilu bio dio nekog određenog softverskog paketa za baze podataka, te se nije mogao koristiti izvan tog paketa (baze).

U današnje vrijeme, aplikacije se najčešće razvijaju u standardnim **objektno orijentiranim** programskim jezicima (Java, C++, ...). Za interakcije s bazom koriste se unaprijed pripremljene klase objekata. Ovakva tehnika je dovoljno produktivna zbog korištenja gotovih klasa, a rezultujući program se lako dotjeruje, integriše u veće sisteme ili prenosi s jedne baze na drugu.

2.5. POZNATI SOFTVERSKI PAKETI ZA RAD S BAZAMA PODATAKA

Baze podataka se u pravilu realizuju korištenjem nekog od provjerenih softverskih paketa. Tabelarni prikaz koji slijedi daje pregled nekih softverskih paketa koji u predstavljaju tehnološki vrh, te imaju značajan udio na svjetskom tržištu.

Gotovo svi današnji softverski paketi podržavaju relacioni model i SQL. Svaki od njih sadrži svoj DBMS, uobičajene klijente (na primjer interaktivni interpreter SQL), te biblioteke i alate za razvoj aplikacija. Svaki paket isporučuje se u verzijama za razne računarske platforme (operativne sisteme).

Konkurencija medu proizvođačima softvera za baze podataka je izuzetno velika, tako da je posljednjih godina često dolazilo do njihovog nestanka, spajanja ili preuzimanja. Lista relevantnih softverskih paketa zato je svake godine sve kraća.

Proizvođač	Proizvod	Operativni sistem	Jezici
IBM Corporation	DB2	Linux, UNIX (razni), MS Windows NT/2000/XP, VMS, MVS, VM, OS/400	SQL, COBOL, Java, . . .
Oracle Corporation	Oracle	MS Windows (razni), Mac OS, UNIX (razni), Linux i drugi	SQL, Java i drugi
IBM Corporation (prije : Informix Software Inc.)	Informix	UNIX (razni), Linux, MS Windows NT/2000/XP	SQL, Java i drugi
Microsoft	MS SQL Server	MS Windows NT/2000/XP	SQL, C++, . . .
MySQL AB	MySQL	Linux, UNIX (razni), MS Windows (razni), Mac OS	SQL, C, PHP, . . .
Sybase Inc.	Sybase SQL Server	MS Windows NT/2000, OS/2, Mac, UNIX (razni), UNIXWare	SQL, COBOL, . . .
Hewlett Packard Co.	Allbase/SQL	UNIX (HP-UX)	SQL, 4GL, C, . . .
Cincom	Supra	MS Windows NT/2000, Linux,	SQL,
Systems Inc.		UNIX (razni), VMS, MVS, VM	COBOL, . . .
Microsoft Corporation	MS Access	MS Windows (razni)	Access Basic, SQL

2.6. ŽIVOTNI CIKLUS BAZE PODATAKA

Uvođenje baze podataka u neko preduzeće ili ustanovu predstavlja složeni zadatak koji zahtijeva timski rad stručnjaka raznih profila. To je projekt koji se može podijeliti u pet faza:

1. analiza potreba,
2. modeliranje podataka,
3. implementacija,
4. testiranje
5. održavanje.

2.6.1. ANALIZA POTREBA

Proučavaju se tokovi informacija u preduzeću. Uočavaju se **podaci** koje treba pohranjivati i veze među njima. U velikim preduzećima, gdje postoje razne grupe korisnika, pojavit će se razni "pogledi" na podatke. Te poglede treba uskladiti tako da se eliminiše redundancija i nekonzistentnost.

Redudantnost podataka *podrazumijeva da se u neki podaci memorisani dva (ili više) puta, ili da se mogu dobiti iz drugih podataka.*
Konzistentnost podataka *obuhvata validnost, tačnost, upotrebljivost i integritet povezanih podataka.*

Analiza potreba također treba obuhvatiti analizu **transakcija** (operacija) koje će se obavljati nad bazom podataka, budući da to može imati uticaja na sadržaj i konačni oblik baze. Važno je procijeniti frekvenciju i opseg pojedinih transakcija, te zahtjeve u vezi sa performansama.

***Performanse** baze podataka obuhvataju skup osobina koje utječu na osobine baze, kao što je brzina izvršavanja operacija obrade podataka, dostupnost podataka, racionalno korištenje resursa sistema (procesora, memorije) itd..*

Rezultat analize je dokument (pisan neformalno u prirodnom jeziku) koji se zove **specifikacija potreba**. Postoje formalne metode analize zahtijeva (analize potreba), ali se one koriste uglavnom na velikim projektima, na kojima radi više ljudi organizovanih u projektne timove.

2.6.2. MODELIRANJE PODATAKA

Različiti pogledi na podatke, otkriveni u fazi analize, sintetiziraju se u jednu cjelinu - globalnu shemu. Precizno se utvrđuju tipovi podataka. Shema se dalje dotjeruje ("normalizira") tako da zadovolji neke zahtjeve kvalitete. Također, shema se prilagođava ograničenjima koje postavlja zadani model podataka, te se dodatno modificira da bi bolje mogla udovoljiti zahtjevima na performanse. Na kraju se iz sheme izvode pogledi (pod-sheme) za pojedine aplikacije (grupe korisnika).

2.6.3. IMPLEMENTACIJA

Na osnovu šeme i podšema, te uz pomoć dostupnog DBMS-a, fizički se realizuje baza podataka na računaru. U DBMS-u obično postoje parametri kojima se može utjecati na fizičku organizaciju baze. Parametri se podešavaju tako da se osigura efikasan rad najvažnijih transakcija. Razvija se skup programa koji realiziraju pojedine transakcije te pokrivaju potrebe raznih aplikacija. Baza se inicijalno puni podacima

2.6.4. TESTIRANJE

Korisnici testiraju bazu i provjeravaju da li ona zadovoljava svim zahtjevima. Nastoje se otkriti greške koje su se mogle potkrasti u svakoj od faza razvoja: dakle u analizi potreba, modeliranju podataka, implementaciji. Greške u ranijim fazama imaju teže posljedice. Na primjer, greška u analizi potreba uzrokuje da transakcije možda korektno rade, no ne ono što korisnicima treba već nešto drugo. Dobro bi bilo kad bi takve propuste otkrili prije implementacije. Zato se u novije vrijeme, prije prave implementacije, razvijaju **prototipovi** baze podataka, te se oni pokazuju korisnicima. Jeftinu izradu prototipova omogućavaju jezici 4. generacije i objektno-orijentisani jezici.

2.6.5. ODRŽAVANJE

Odvija se u vrijeme kad je baza već ušla u redovnu upotrebu (produktivna baza). Sastoji se od sljedećeg:

- popravak grešaka koje nisu bile otkrivene u fazi testiranja
- uvođenje promjena zbog novih zahtjeva korisnika
- podešavanje parametara u DBMS u svrhu poboljšavanja performansi.

Održavanje zahtijeva da se stalno prati rad s bazom, i to tako da to praćenje ne ometa korisnike. Administratoru baze podataka trebaju stajati na raspolaganju odgovarajući alati (*utility* programi).

2.7. MODEL OBJEKTI-VEZE

2.7.1. OSOBINE O-V (E-R) MODELA

Model objekti-veze je konceptualni model koji realni svijet vidi kao kolekciju objekata (entiteta) i veza medu njima. Osnovna komponenta ovog modela je dijagram objekti veze (engl. *Entity-Relationship diagram*) koji se koristi za vizuelno predstavljanje objekata.

Model objekti-veze omogućava:

- **Jednostavno preslikavanje (prevođenje) u relacioni model;** kao što je već napomenuto, relacioni model koristimo u slijedećoj fazi (logički dizajn); Relacioni model je jedan od najčešće korištenih modela za koji danas postoji čitav niz sistema za upravljanje bazama podataka u okviru kojih bazu podataka možemo izgraditi i u okviru kojeg je možemo koristiti
- Model je ***jednostavan***, lako se uči i ne traži posebno obučen kadar. Stoga ga dizajneri mogu koristiti za komunikaciju sa krajnjim korisnikom čija je pomoć neophodna u toku izrade baze podataka.
- Model se može lako koristiti kao ***osnova za implementaciju*** u okruženju odabranog sistema za upravljanje bazama podataka koji se danas nude na tržištu.

2.7.2. MODELIRANJE ENTITETA I VEZA

Bavimo se pitanjem: kako oblikovati shemu za bazu podataka, usklađenu s pravilima relacionog modela. U stvarnim situacijama dosta je teško direktno dizajnirati relacionu shemu. Zato se služimo pomoćnom fazom koja se zove **modeliranje entiteta i veza** (engl. *Entity-Relationship Modelling*). Riječ je o oblikovanju jedne manje precizne, konceptualne sheme, koja predstavlja apstrakciju realnog svijeta. Ta tzv. ER-shema se dalje, više-manje automatski, pretvara u relacionu. Modeliranje entiteta i veza zahtijeva da se svijet promatra preko tri kategorije:

- **Entiteti (*entity*):** objekti ili događaji koji su nam od interesa;
- **Veze (*Relationship*):** odnosi među entitetima koji su nam od interesa;
- **atributi:** svojstva entiteta i veza koja su nam od interesa.

2.7.2.1. ENTITETI I ATRIBUTI

Entitet je nešto o čemu želimo memorisati podatke, nešto što je u stanju postojati ili ne postojati, te se može identifikovati. Entitet može biti objekt ili biće (na primjer kuća, student, auto), odnosno događaj ili pojava (na primjer nogometna utakmica, praznik, servisiranje auta).

Entitet je opisan **atributima** (na primjer atributi kuće su: adresa, broj katova, boja fasade, . . .). Ukoliko neki atribut i sam zahtijeva svoje atribute, tada ga radije treba smatrati novim entitetom (na primjer model auta). Isto pravilo vrijedi i ako atribut može istovremeno imati više vrijednosti (na primjer kvar koji je popravljen pri servisiranju auta).

Ime entiteta, zajedno sa pripadajućim atributima, zapravo određuje **tip** entiteta. Može postojati mnogo **primjeraka** (pojava) entiteta zadanog tipa (na primjer *UČENIK* je tip čiji primjerci su Šabić Mirza, Marković Marko, . . .).

Kandidat za ključ je atribut, ili skup atributa, čije vrijednosti jednoznačno određuju primjerak entiteta zadanog tipa. Dakle, ne mogu postojati dva različita primjerka entiteta istog tipa s istim vrijednostima kandidata za ključ. Na primjer, za tip entiteta *AUTO*, kandidat za ključ je atribut *REG BROJ*. Ukoliko jedan tip entiteta ima više kandidata za ključ, tada biramo jednog od njih i proglašavamo ga **primarnim ključem**. Na primjer, primarni ključ za tip entiteta *STUDENT* mogao bi biti atribut *BROJ INDEKSA*.

2.7.2.2. VEZE

Veze se uspostavljaju između dva ili više tipova entiteta (na primjer veza *IGRA ZA* između tipova entiteta *IGRAC* i *TIM*). Zapravo je riječ o imenovanoj binarnoj ili *k*-narnoj relaciji između primjeraka entiteta zadanih tipova. Za sada ćemo se ograničiti na veze između tačno dva tipa entiteta.

Funkcionalnost veze može biti:

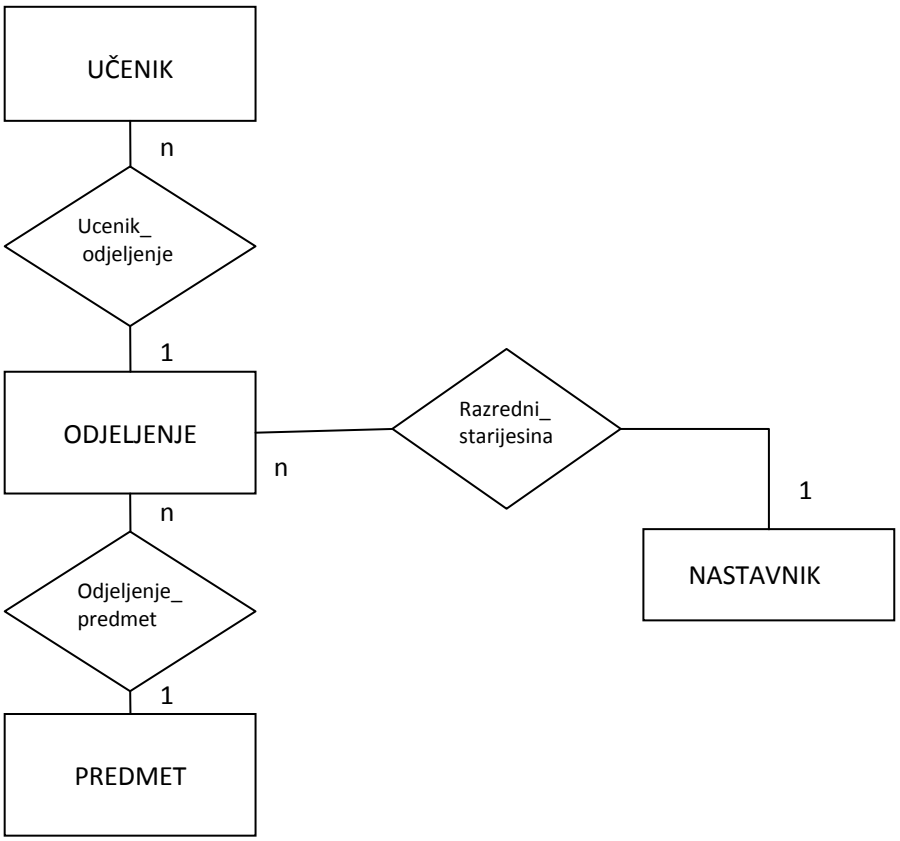
- **Jedan-naprema-jedan** (1 : 1, *one-to-many*)**.** Jedan primjerak prvog tipa entiteta može biti u vezi s najviše jednim primjerkom drugog tipa entiteta, te također jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkom prvog tipa. Na primjer veza *JE RAZREDNIK* između tipova entiteta N*ASTAVNIK* i *ODJELJENJE*.
- **Jedan-naprema-više** (1 : *N, one-to-many*)**.** Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjeraka drugog tipa entiteta, no jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkom prvog tipa. Na primjer veza *ucenik_odjeljenje* između tipova entiteta *UCENIK i ODJELJENJE* .
- **Više-naprema-više** (*M* : *N, many-to-many*)**.** Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjeraka drugog tipa entiteta, te također jedan primjerak drugog tipa može biti u vezi s 0, 1 ili više primjeraka prvog tipa. Na primjer veza *predmet_odjeljenje* između tipova entiteta *PREDMET* i *ODJELJENJE* .

Veza može imati i svoje atribute koje ne možemo pripisati ni jednom od tipova entiteta (na primjer veza *učenik_odjeljenje* može imati atribut *DATUM UPISA*).

Ako svaki primjerak entiteta nekog tipa mora učestvovati u zadanoj vezi, tada kažemo da tip entiteta ima **obavezno članstvo** u toj vezi. Kažemo da je veza jaka. Inače tip entiteta ima neobavezno članstvo (slaba veza). Na primjer, između tipova entiteta *UCENIK* i *ODJELJENJE* zadana je veza, koja ima funkcionalnost (N : 1). To je jaka veza, jer učenik mora biti u nekom odjeljenju.

2.7.3. PRIKAZ ER-SHEME POMOĆU DIJAGRAMA (ER DIJAGRAM)

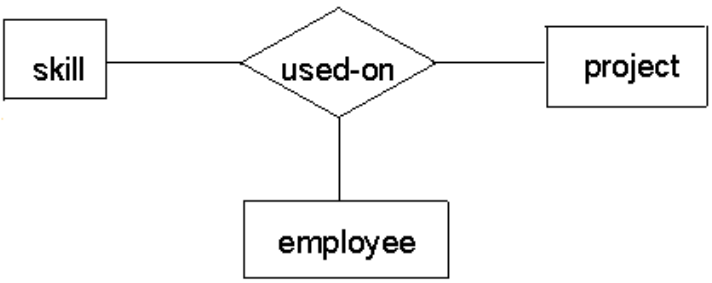
Uobičajeno je da se ER-shema nacrta kao dijagram u kojem pravougaonici predstavljaju tipove entiteta, a rombovi veze. Veze su povezane linijama s odgovarajućim tipovima entiteta. Imena tipova entiteta i veza, te funkcionalnost veza, uneseni su u dijagram. Posebno se prilaže lista atributa za svaki entitet odnosno vezu. U toj listi možemo specificirati obaveznost članstva u vezama.



Slika 19. Primjer ER dijagrama

2.7.4. SLOŽENIJE VEZE

Ternarne veze uspostavljaju se između tri tipa entiteta. Znači riječ je o ternarnoj relaciji između primjeraka triju tipova entiteta. Postoje brojne mogućnosti za funkcionalnost ternarne veze, na primjer (*N* : *M* : *P*), (1 : *N* : *M*), (1 : 1 : *N*) ili čak (1 : 1 : 1).



Slika 20. Primjer ternarne veze

Primjer ternarne veze sa prethodne slike odnosi se na podatke o vještinama (SKILL), projektima (PROJECT) i zaposlenim (EMPLOYEE) angažovanim na projektima. Funkcionalnost ove veze je mnogo-naprema-mnogo-naprema-mnogo, dakle (*N* : *M* : *P*), jer na primjer za zadani par (projekat, vještina) potrebno je mnogo zaposlenih, itd..

ER model dovoljno je jednostavan da ga ljudi različitih struka mogu razumjeti. Zato ER shema služi za komunikaciju projektanta baze podataka i korisnika, i to u najranijoj fazi razvoja baze. Postojeći DBMS ne mogu direktno implementirati ER shemu, već zahtijevaju da se ona detaljnije razradi, te modificira u skladu s pravilima relacijskog modela.

2.8. RELACIONI MODEL

Objekte i veze u relacionom modelu podataka predstavljamo relacijama.

Relacija u relacionom modelu podataka isto je što i relacija u matematici, s tim što su relacije u relacionom modelu vremenski promjenljive. Relacioni model podataka teorijski je razradio britanski matematičar Codd E.F. Codd-ova definicija relacije glasi:

Neka su dati skupovi D1, D2, D3...Dn (ne obavezno različiti). R je *relacija* nad ovih n skupova (n veće od 0) ako je to skup n-torki takav da za svaku n-torku vrijedi da je prvi element n-torke iz D1, drugi iz D2.... n-ti element iz Dn.

Skup D1 naziva se *domena* relacije R. Domen je skup sličnoga tipa, npr. skup svih prezimena učenika. Domen je, dakle, skup svih vrijednosti iz kojeg neki element n-torke relacije može uzeti vrijednost. Uobičajen zapis relacije je:

Naziv_relacije (atribut1, atribut2,....,atribut n)

Npr. relacija učenik zapisana je kao:

Ucenik(#ucenik, prezime, ime, adresa)

Primjerak relacije je konkretizacija relacije. Na primjer.

(1,“Jahić“,“Adnan“,“Armije BiH 18“)
(2,“Mujić“,“Damir“,“Albina Herljevića 9“)
(3,“Tomić“,“Franjo“,“Bizovac 45“)

S aspekta korisnika, relacija je dvodimenzionalna tabela, koju nazivamo *relacionom tabelom*. Na primjer

#ucenik	Prezime	Ime	Adresa
1	Jahić	Adnan	Armije BiH 18
2	Mujić	Damir	Albina Herljevića 9
3	Tomić	Franjo	Bizovac 45

Relaciona tabela naziva se i tabela (*table*). Atributi tabele nazivaju se i polja (engl. *fields*). Primjerak relacije naziva se i slog (engl. *record*).

Skup svih tabela koje čine relacionu bazu podataka naziva se šema *baze podataka (schema).*

Kolone relacione tabele su atributi relacije. Svaki red relacione tabele predstavlja jednu n-torku relacije ili primjerak relacije.

2.8.1. SVOJSTVA RELACIONE TABELE

Relaciona tabela ima slijedeća **svojstva**:

- Vrijednosti su atomične
  - kolone (atributi) su nedjeljive informacije, tj. ne mogu biti složeni iz više podataka.
  - npr, tabela razred ne može imati u jednoj koloni atribute razred i odjeljenje; razred i odjeljenje moraju biti odvojene kolone
- Sve vrijednosti jedne kolone su istog tipa
  - ovo znači da je domen podataka u jednoj koloni isti (domen je skup vrijednosti koje atribut može poprimiti)
  - npr, za sve primjerke relacione tabele Predmet kolona „oblik nastave“ može poprimiti samo vrijednosti iz skupa ("teroretska", "prakticna", "laboratorijska", "kombinovano"). U ovoj koloni ne može biti upisana neka druga vrijednost.

- Svaki red tabele je jedinstven
    - Ovo svojstvo obezbjeđuje da u relacionoj tabeli ne postoje dva identična reda; postoji najmanje jedna kolona, ili skup kolona, čija vrijednost jedinistveno identifikuje svaki red relacione tabele. Ova kolona (kolone) nazivamo primarnim ključem.
    - Npr. u relacionoj tabeli Predmet dozvoljeno je postojanje dva reda sa identičnim vrijednostima atributa: naziv_predmeta, razred, oblik i sedmicni_fond, ali ako atribut predmet_id ima različite vrijednosti.
- Redoslijed kolona je nevažan.
- Redoslijed redova je nevažan.
- Svaka kolona mora imati jedinstveno ime.

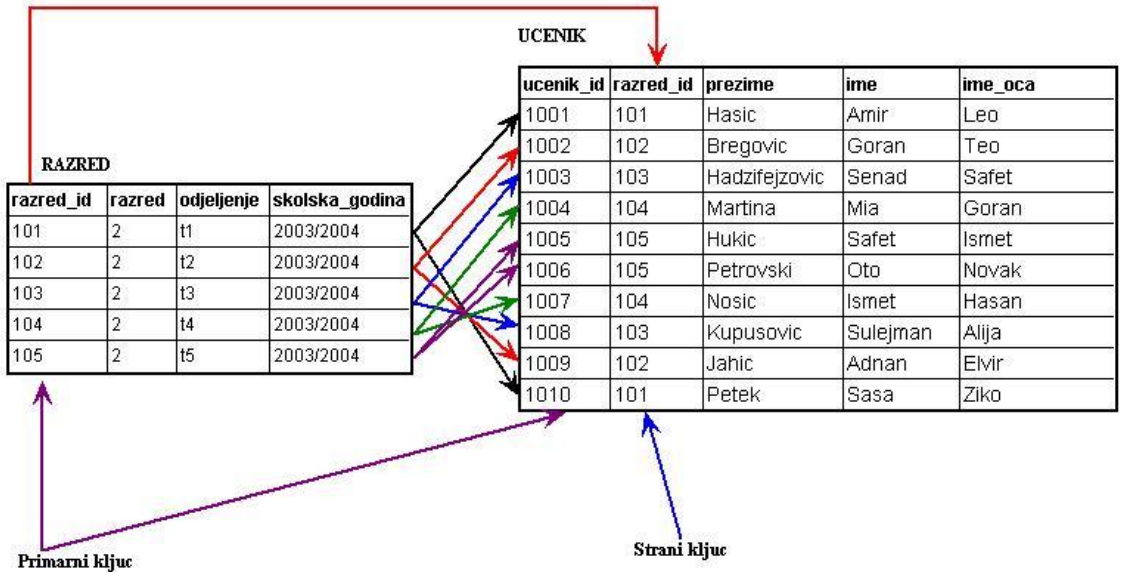
U relacionom modelu postoje samo relacije ili relacione tabele. Odnos između dvije ili više tabela izražen je pomoću vrijednosti kolona koje predstavljaju primarni i strani ključ.

2.8.2. PRIMARNI I STRANI KLJUČ (PK I FK)

Primarni ključ je jedna ili više kolona tabele čija vrijednost jedinstveno identifikuje svaki red tabele.

Strani ključ je jedna ili više kolona čije vrijednosti su iste kao i vrijednosti primarnog ključa druge tabele. O stranom ključu možemo razmišljati kao o kopiji primarnog ključa neke druge tabele. Veza između dvije relacione tabele uspostavlja se preko identifikovanja primjeraka čija je vrijednosti stranog ključa u jednoj tabeli jednaka vrijednosti primarnog ključa druge tabele.

Posmatrajmo relacije Razred i Ucenik. U relacionoj tabeli *Ucenik* atribut *rezred_id* je strani ključ. Veza između tabela *Razred* i *Ucenik* uspostavljena je na taj način što se identifikuje iste vrijednosti primarnog ključa tabele Razred (razred_id) i stranog ključa "razred_id" u tabeli Ucenik.



Slika 21. Primjeri primarnog i stranog ključa unutar tabela

2.8.3. PRESLIKAVANJE MODELA OBJEKTI-VEZE U RELACIONI MODEL

(Pravila za prevođenje modela objekti-veze u relacioni model)

Model objekti-veze koji je napravljen u prethodnoj fazi razvoja baze podataka, jednostavno se prevodi u relacioni model na osnovu skupa pravila. Nabrojati ćemo pravila, pokazati na primjeru njihovu primjenu i generisati relacionu šemu koja i jeste cilj ovog koraka razvoja baze podataka.

Najprije se primjenjuje jednostavno pravilo za prevođenje objekata u relacione tabele. To pravilo glasi:

Pravilo 1: Prevođenje objekata

Svaki objekat modela objekti veze postaje relaciona tabela, i to tako da atributi objekta postaju atributi relacione tabele, primarni ključ objekta postaje primarni ključ relacione tabele.

Nakon prevođenja objekata u relacione tabele, prevodimo veze primjenom pravila za prevođenje veza. Pravila za prevođenje veza date su na osnovu ***kardinalnosti veze i sveobuhvatnosti objekata u vez***i.

Prilikom izgradnje modela objekti veze identifikovali smo kardinalnost veze (u osnovi tri tipa veza: 1:1, 1:N i N:M). Kada prevodimo model objekti-veze u relacioni model, moramo voditi računa o još jednom svojstvu veze, a to je količina objekata koji učestvuju u vezi - naziva se još i sveobuhvatnost objekata u vezi. S tim u vezi, razlikujemo parcijalnu i totalnu vezu.

- Veza je **totalna** za objekat ako svi objekti tog tipa učestvuju u bar jednom pojavljivanju veze.
  - Npr, veza Razred-Ucenik je totalna s obje strane , jer ako posmatramo primjerke objekta Razred, možemo uočiti da svaki primjerak razreda učestvuje u više veza sa Učenikom (nema razreda koji nema učenika); s druge strane, svaki primjerak ucenika učestvuje u tačno jednoj vezi sa razredom (nema učenika koji ne ide u neki razred). S toga, oba objekta su totalna u ovoj vezi.
- Veza je **parcijalna** za objekat ako svi objekti tog tipa ne učestvuju u pojavljivanju veze.
  - Npr. veza Razredni_starjesina_Razred je parcijalna sa strane Profesor, a totalna sa strane Razred. Naime, postoje primjerci Profesora koji nisu u relaciji Razredni_strajesina ni sa jednim primjerkom Razreda, jer uopšte nemaju razredno starješinstvo. S druge strane, ne postoji primjerak Razreda a da nema vezu sa primjerkom Profesor, jer svaki razred ima razrednog starješinu.

Možemo modifikovati dijagram objekti-veze da bismo dodali i informaciju o sveobuhvatnosti veze. Ako objekat učestvuje totalno u vezi, onda uz informaciju o kardinalnosti, uz objekat upisujemo i znak *.

Dakle, uzimajući u obzir kardinalnost veze i sveobuhvatnost objekata u vezi, primjenjujemo slijedeća pravila za prevođenje veza:

Pravilo 2: prevođenje veze 1:N kod koje je strana N totalna

Svaki 1:N tip veze gdje je strana N totalna (svaki primjerak objekta na strani n je povezan sa jednim objektom na strani 1) ne prevodi se u novu relaciju, nego se primarni ključ objekta sa strane 1 umeće kao atribut u relacionu tabelu na strani N. Ovaj atribut postaje strani ključ relacione tabele na strani N. Svi eventualni artibuti veze postaju atributi relacije na strani N.

Pravilo 3: prevođenje veze 1:N kod koje je strana N parcijalna

Svaki 1:N tip veze gdje je strana N parcijalna (postoje primjerci objekta na strani n koji nisu povezani sa objektom na strani 1) posmatra se kao veza M:N i prevodi po pravilu 4. Svi eventualni atributi veze postaju atributi nove relacione tabele.

Pravilo 4: prevođenje veze N:M

Svaka N:M tip veze postaje nova relaciona tabela, atributi tipa veze postaju atributi nove relacione tabele, primarni ključ nove relacione tabele je složen od primarnih ključeva objekata koji učestvuju u vezi.

Pravilo 5: prevođenje veze 1:1 koja je s obje strane totalna

Svaka veza 1:1 koja je s obje strane totalna prevodimo u samo jednu relacionu tabelu, i to tako da svi atributi objekata i atributi veze postaju atributi nove relacione tabele, a primarni ključ čine primarni ključevi objekata obuhvaćenih vezom.

Pravilo 6: prevođenje veze 1:1 koja je s jedne strane parcijalna, a sa druge totalna

Svaka 1:1 veza koja je s jedne strane parcijalna, a sa druge totalna prevodi se tako da atributi veze postaju atributi relacione table s totalne strane, a primarni ključ relacije s parcijalne strane postaje novi atribut u relacionoj tabeli sa totalne strane; ovaj novi atribut je strani ključ u relacionoj tabeli sa totalne strane.

Pravilo 7: prevođenje veze koja obuhvata više od dva objekta

Ako su vezom obuhvaćena više od dva objekta, pri čemu je kardinalnost veze M, uvodi se nova relaciona tabela, čiji ključ je složen od primarnih ključeva svih objekata obuhvaćenih vezom. Ukoliko veza ima atributa, oni postaju kolone nove relacione tabele.

2.9. MS ACCESS

2.9.1. NAMJENA I OSOBINE PROGRAMSKOG PAKETA MS ACCESS

Access je sistem za upravljanje relacionim bazama podataka (engl. *Relation Database Management System, RDBMS*). Dolazi u paketu Microsoft Office, zajedno sa ostalim aplikacijama (*Excel, Word, PowerPoint, Outlook*). Access skladišti i učitava podatke, prikazuje podatke i automatizira mnoge poslove. Pomoću Accessa možete da razvijete obrasce (*Forms*) za unošenje podataka koji se lako koriste. Možete da obrađujete podatke i sastavljate složene izvještaje (*Reports*). Microsoft Access spaja produktivnost programa za upravljanje bazama podataka sa lakoćom korištenja Windowsa.

Pomoću OLE (engl. *Object Linking and Embending* – povezivanje i ugrađivanje objekata) objekata koje koriste u Windowsu i drugim proizvodima iz Microsoftovog paketa Office možete da proširite Access u pravo okruženje za rad sa bazama podataka tako što ćete ga integrisati sa tim proizvodima.

Access možete da vežete sa podacima koji se nalaze na centralnom računaru ili serveru, ili da koristi tabele koje su kreirane u dBASE-u ili Excelu. Rezultate obrade lako možete da prenesete na radni list Excela.

Access obezbjeđuje potpuno upravljanje bazama podataka, obezbjeđuje primarne ključeve i referencijalni integritet. Poljima u Accessu mogu da se pridruže pravila za ispravnost podataka koja sprečava unošenje neispravnih podataka bez obzira na način na koji se oni unose. Svako polje tabele ima definicije za format i podrazumijevanu vrijednost, što obezbjeđuje produktivniji unos.

2.9.2. OBJEKTI ACCESS BAZE PODATAKA

Generalno baza podataka jeste fizički fajl na disku ili više njih u zavisnosti od toga kako je RDBMS konfigurisan. Međutim, pored toga svaka baza ima još jedan sloj (logički) unutar fizičke strukture koji služe za pohranu i manipulaciju sa podacima. To su *objekti baze podataka*.

Svaki RDBMS ima pojedine specifične elemente, ali generalno ovo su zajednički za sve sisteme:

- Tabele
- Pogledi (*Views*);
- Trigeri
- Procedure.

U odnosu na navedenu listu, Access se u nekim elementima razlikuje, dok je negdje u pitanju imenovanja objekata:

- **Tabele** – osnovni gradivni element svake baze podataka (služe kao skladišta za podataka) Funkcije i izrazi - za provjeru ispravnosti unesenih podataka i eventualni proračuni nad njima (npr. provjera je li uneseni podatak datum, broj ili izračunava platu radnika na osnovu ulaznih podataka iz jedne ili više tabela)
- **Makro naredbe** - set unaprijed definisanih automatskih operacija koje se mogu implementirati bez programiranja
- **VBA** – *Visual Basic for Aplications* ugrađeni jezik za programiranje unutar Access-a.
- **API** – mogućnost korištenja API poziva iz VBA koda baze u Access-u.

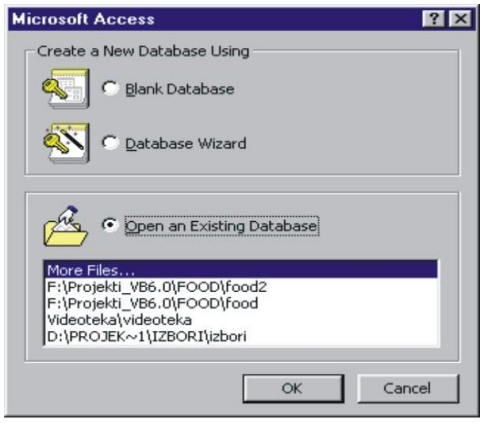
Slika 22. Hijerarhija elemenata u Access-u

Ako navedene komponente pokušamo posmatrati sa strane korisnika, tada dobijamo sljedeću strukturu i raspored elemenata.

Slika 23. Postupak rada sa bazom sa aspekta korisnika

Međutim, sa aspekta dizajnera baze podataka i implementacije sistema proces rada je sasvim drugačiji. Preko 70% posla je nevidljivo za krajnjeg korisnika tj. korisnički interfejs je samo jedan mali dio onoga što se nalazi u ostatku sistema.

2.9.3. RAD SA ACCESS-OM

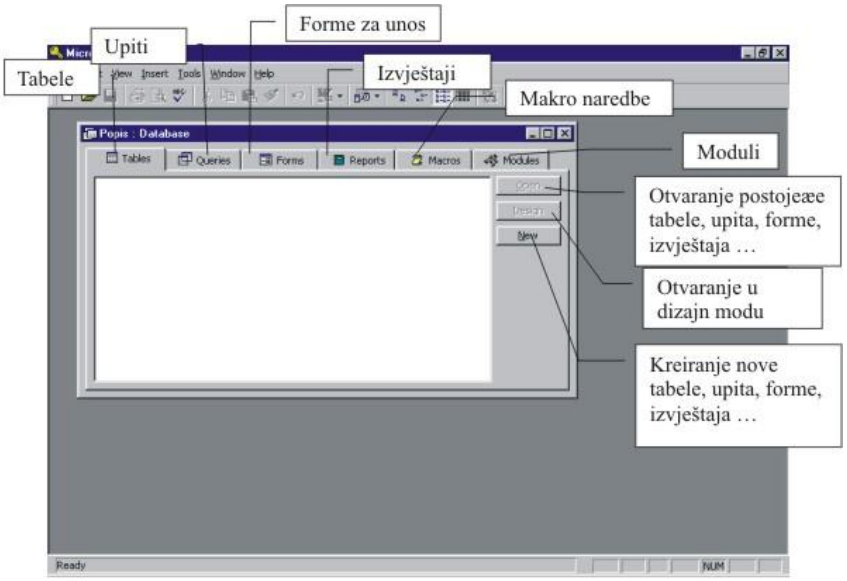


2.9.3.1. ULAZNA MASKA U ACCESS

Prilikom pokretanja Access-a mogu se izabrati tri opcije

- Otvoriti praznu bazu (Blank database). Ako se izabere ova opcija baza se odmah snimi na disk pod određenim imenom
- Pokrenuti "čarobnjaka" za baze, koji će pomoći na da kreiranju baze podataka (Database Wizard),
- Otvoriti postojeću bazu (Open an Existing Database)

Izgled radnog okruženja MS Access 2000 je prikazan na sljedećoj slici



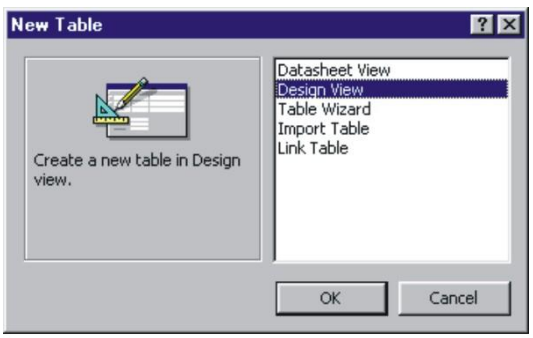
2.9.3.2. TIPOVI POLJA (PODATAKA) U ACCESS-U

- Tekstualno (Text) – Alfanumerički znaci; najviše 255 znakova,
- Brojčano (Number) – brojčani tipovi (Integer, Long, Double …)
- Automatski brojač (AutoNumber) – Brojač koji se automatski povećava
- Novčano (Currency) – Valute
- Datum/Vrijeme (Date/Time) – Datum i vrijeme
- Memo (Memo) - Alfanumerički znaci: dugački nizovi do 64000 znakova,
- Logičko (Yes/No) – Logičke vrijednosti (0 i 1, Da i Ne, True i False)
- Hiperveza (HyperLink) – Veza sa sadržajima na računaru ili na Internetu
- OLE objekat – Slike, dijagrami, zvučni zapisi, video zapisi…

Prilikom definisanja tabele za svako polje se mora definisati tip podatka koji će kasnije biti unošen u tabelu.

2.9.3.3. KREIRANJE TABELE

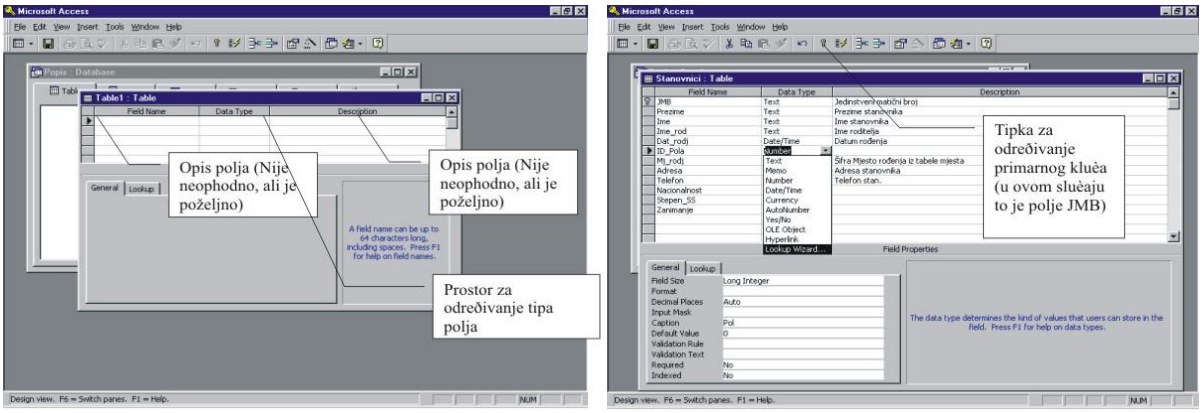
Da bi kreirali tabelu potrebno je se postaviti na karticu *Tables* i pokrenuti komandni taster *New*. Pojaviti će se sljedeći *dialog box*:



Nakon toga mogu se iskoristiti sljedeće opcije za kreiranje tabele:

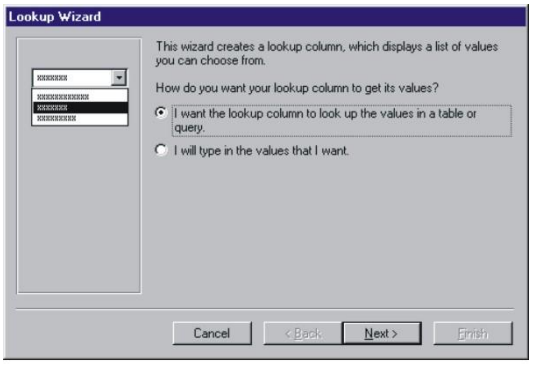
- Datasheet view (Slično kao u Excel-u)
- **Design View**
- Table Wizard (Korištenje pomoćnika – "čarobnjaka")
- Import Table (uvoženje) tabele iz neke postojeće baze
- Link Table – povezivanje postojeće tabele iz postojeće baze

Uglavnom se koristi Design View.

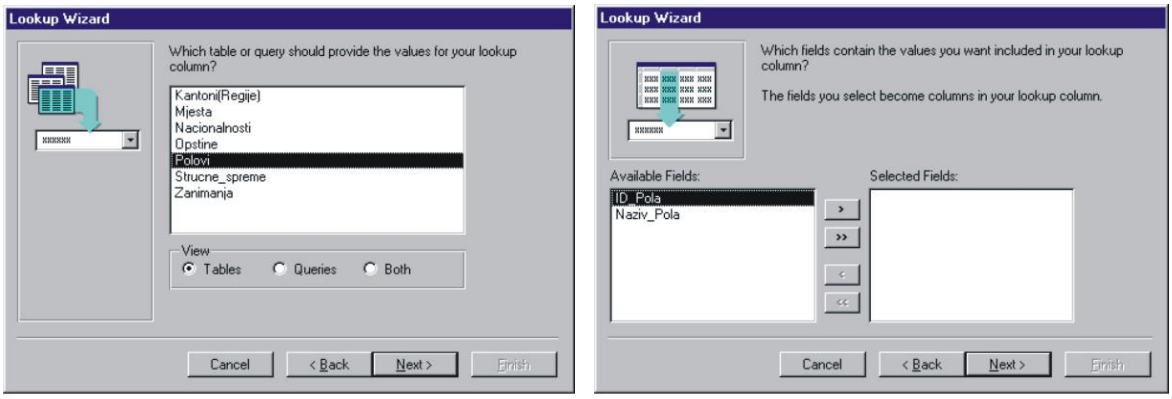


Nakon definisanja svih polja tabela se snimi pod odgovarajućim imenom. Prilikom definisanja imena poželjno je da ona sadržavaju samo slova engleskog alfabeta, brojeve i znak _ (bez praznih mjesta).

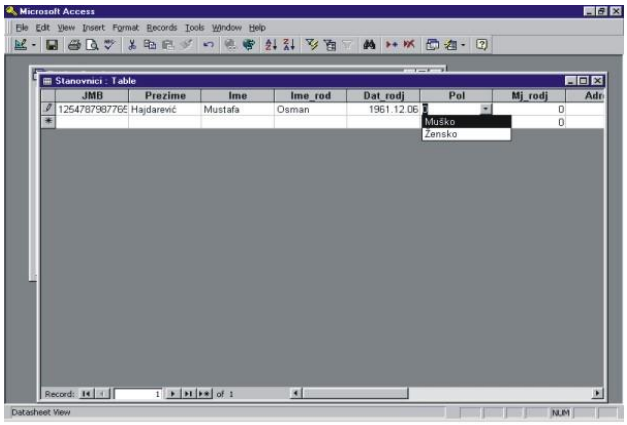
Access je „user friendly" i u svemu pokušava pomoći korisniku. Tako je i sa povezivanjem tabela. Ako između dvije tabele postoji odnos ***n – 1*** preko nekog polja dovoljno je da u tabeli iz koje ide odnos ***n*** u polju gdje se unose pripadajuće vrijednosti iz vezane (u koju ide odnos „naprema 1") tabele pokrene *Lookup Wizard*.



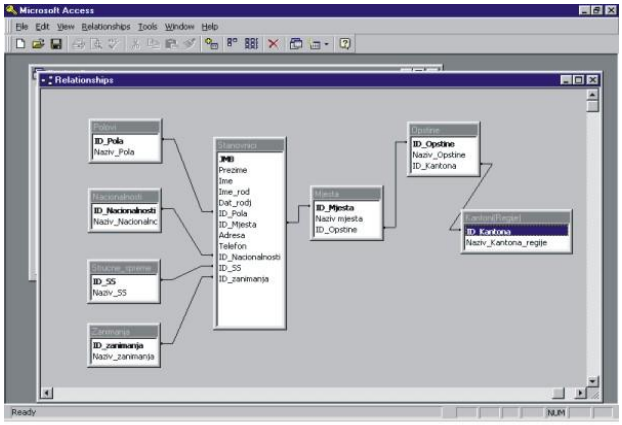
Odabira se opcija *"I want lookup column to look up the values in a table or query"*. Time se Access-u kaže da se želi izvršiti povezivanje tabela i da se žele pregledati vrijednosti koje se unose.



Nakon toga se odabiru polja čiji sadržaj se želi pregledati prilikom unosa podataka, čime je olakšan unos podataka. Na sljedećoj slici je prikazan unos podatka u polje koje je povezano na opisani način.



Ako se na sličan način kreiraju i ostale veze između tabela baza podataka odnosno veze između tabela bi izgledale kao na sljedećoj slici. Ako se koristi *Lookup Wizard* desit će se da Access izjednači imena polja koja su povezana.



2.9.3.4. UPITI (QUERIES)

Engleska riječ *query* potiče od latinske riječi *quaerere*, što znači pitati ili raspitivati se. Prema tome riječ *query* (upit) se može smatrati kao pitanje postavljeno bazi podataka koje se odnosi na podatke smještene u njenim tabelama. Upit može da bude jednostavno pitanje o podacima koji su smješteni u jednoj tabeli, ili složeno pitanje koje se odnosi na podatke smještene u više tabela. Pošto se kreira i pokrene upit Access prikazuje skup zapisa koji su traženi u obliku tabelarnog zapisa. Upiti još služe i kao izvor podataka za izvještaje (*Reports*).

Vrste upita

Najvažniji tipovi upita:

***Select** (izdvajanje podataka)*

- To je najčešća vrsta upita. Upiti za izdvajanje podataka pronalaze i izdvajaju podatke iz jedne ili više tabela (na osnovu uslova koji se zadaju)

***Total** (Zbirni) upiti*

- To je posebna vrsta upita za izdvajanje podataka. Zbirni upiti omogućavaju izračunavanje suma, prosječnih vrijednosti, maksimuma itd. Kada koristite ovaj tip upita, Access dodaje red *Total*.

***Action** (Akcioni)*

- Ovi upiti omogućavaju automatsko kreiranje novih tabela (*Make Tables*), ili ažuriranje (unos, brisanje i ispravka) podataka (brisanje, mijenjanje, dodavanje) u postojećim tabelama.

Upiti se kreiraju pomoću posebnih alata za sastavljanje upita (engl. *queries tools*). Ti alati u stvari kreiraju SQL (*Structured Query Language*) zapis koji se može pregledati i izmijeniti (*Query View*) i koristiti u nekim aplikacijama (npr. Visual Basic).

Šta omogućavaju upiti?

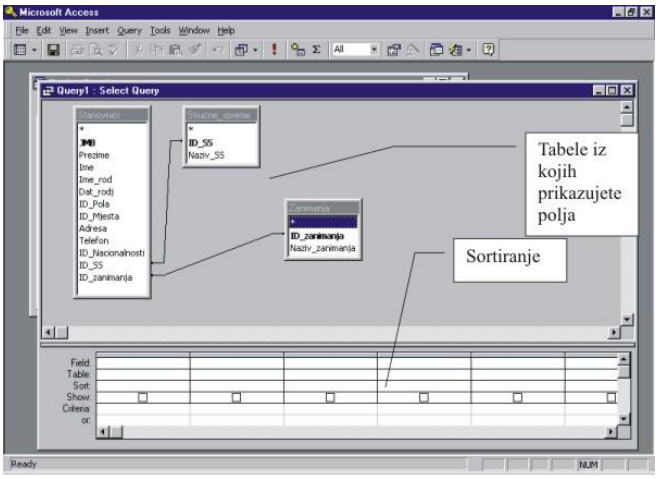
- Izdvajanje podataka
- Sortiranje zapisa
- Proračunavanja
- Kreiranje tabela (Create Table)
- Kreiranje obrazaca (formi) i izvještaja na osnovu upita
- Kreiranje dijagrama na osnovu upita
- Korištenje upita kao izvora podataka za druge upite (podupit)
- Mijenjanje sadržaja tabela

2.9.3.5. KREIRANJE UPITA

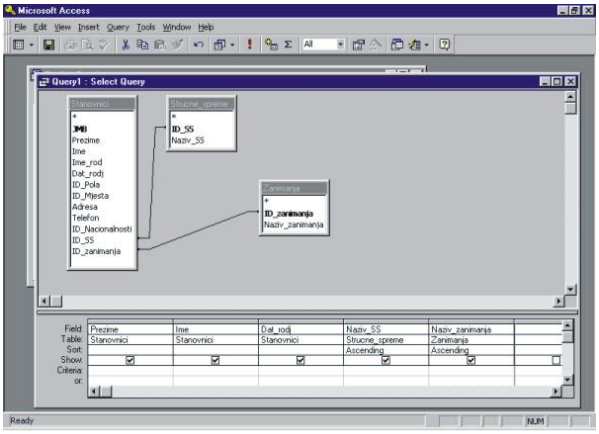
Pošto se kreiraju i popune tabele, mogu se kreirati upite (odnosi se na SELECT upite*) Da bi se kreirao upit potrebno je postaviti se na karticu *Queries* (u prozoru baze podataka) i odabrati dugme *New*. Nakon toga se obično izabere opcija *Design View*.

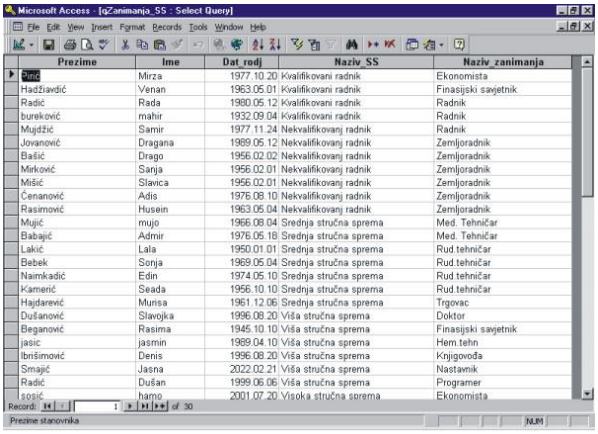
- Nakon toga će se pojaviti okvir sa popisom tabela iz baze na osnovu kojih se može formirati upit.
- Tabele na osnovu kojih se kreira upit moraju se dodati (dugme *Add* ili dvostruki klik na ime tabele).

Podaci se mogu i sortirati po nekom polju ili više njih. Za umetanje polja u *query* koristi se metoda „povuci i ispusti“ (engl. *Drag and Drop*) ili dvostruki klik mišem.



- Nakon dizajniranja, upit se može snimiti.

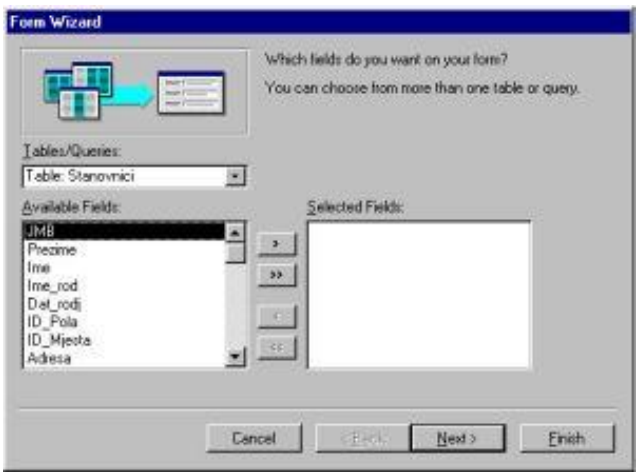


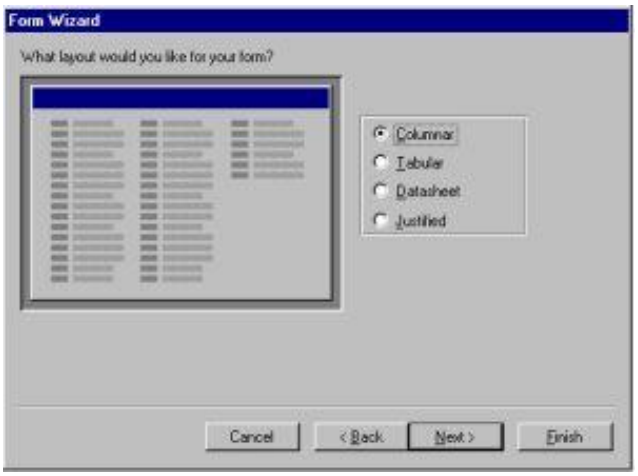


2.9.3.6. FORME ZA UNOS (FORMS)

Access omogućava kreiranje čitave aplikacije. Dio te aplikacije su i forme. Iako se za većinu opcija mogu koristiti “Wizards”, za naprednije opcije, osim upotrebe čarobnjaka (engl. *wizard*), potrebno je poznavati i programiranje u Visual Basic-u.

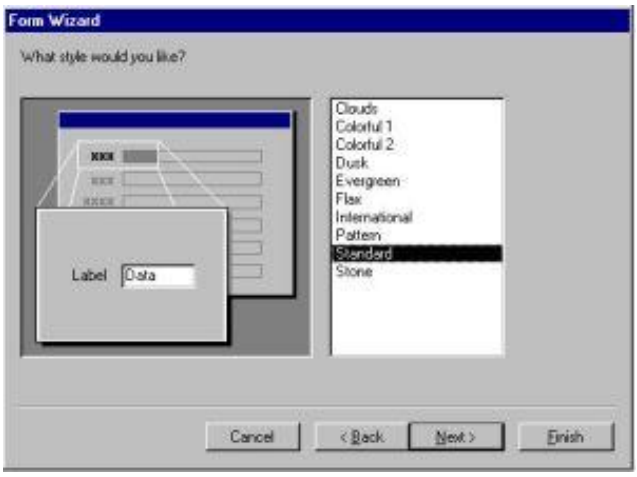
Form wizard

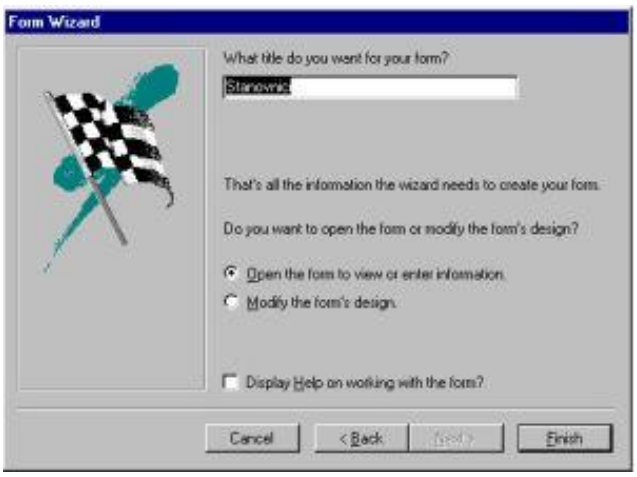




U prvom koraku se izaberu polja koja se žele ažurirati.

- U drugom koraku se definiše izgled i tip forme.
- U trećem koraku se izabere stil





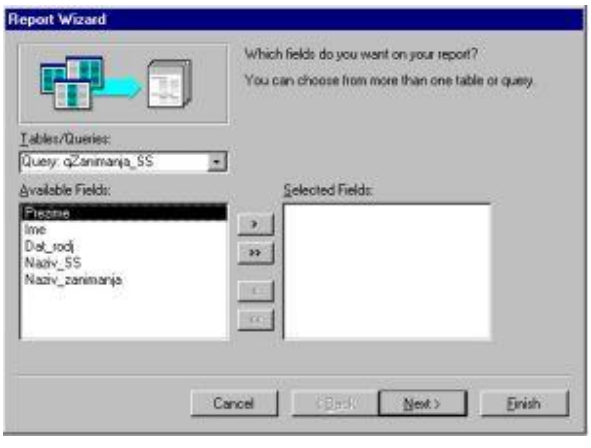
- U zadnjem koraku se definiše naslov forme

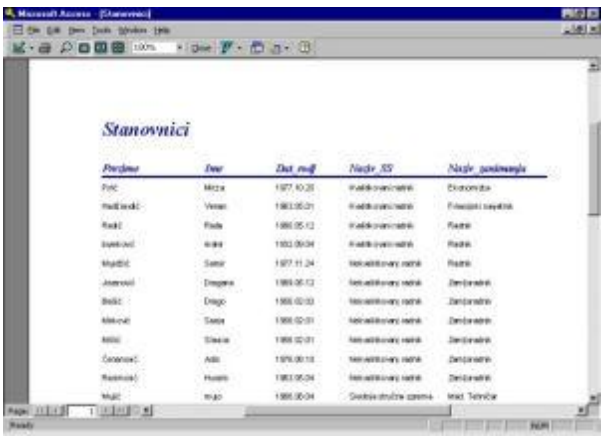
Naknadno se forma može modifikovati (dizajn i programski kod).

2.9.3.7. IZVJEŠTAJI (REPORTS)

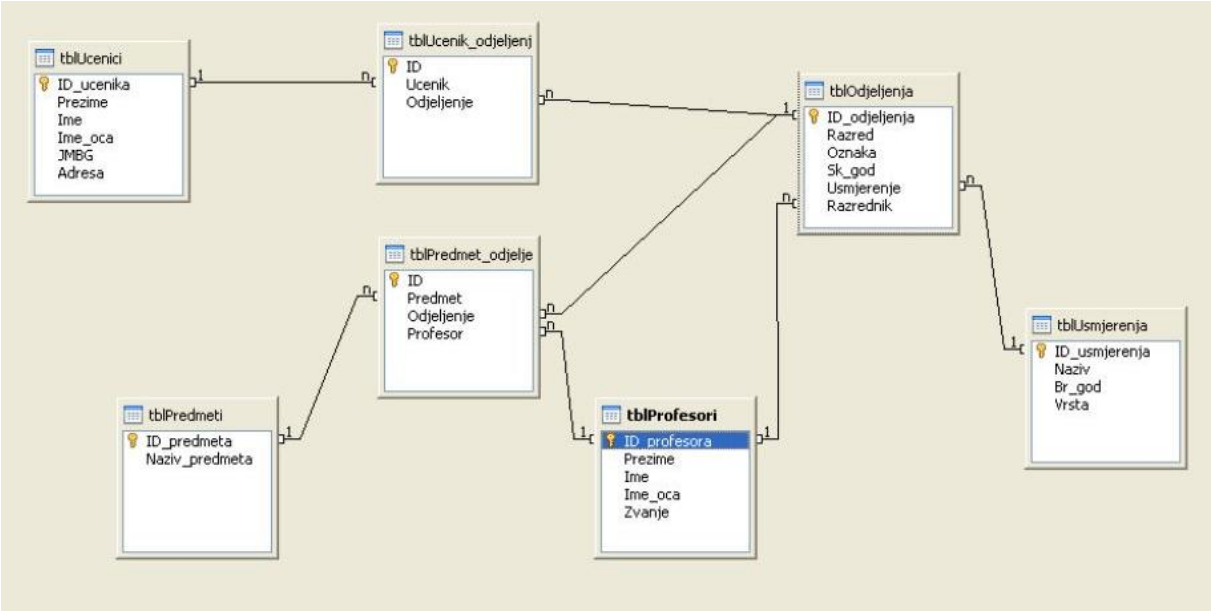
U Accessu se mogu kreirati i izvještaje koji se mogu pregledati i štampati. Izvori podataka za izvještaje su tabele ili upiti. Izvještaji se najčešće kreiraju pomoću pomoćnika (wizard-a).

- Prvi korak je odabiranje izvora za izvještaj.
- Drugi korak je odabir polja koja se prikazuju.





Kreirani izvještaj se može štampati ili snimiti, odnosno izvesti u neki od standardnih formata (tekstualni fajl, Excel datoteka, itd..)



Slika 24. Primjer baze podataka u MS Access-u.

3. PROGRAMSKI JEZIK C++

3.1. PROGRAMSKI JEZICI

Prvi računari bila su vrlo složeni za korištenje. Njih su koristili isključivo stručnjaci koji su bili osposobljeni za komunikaciju s računarom. Ta komunikacija se sastojala od dva osnovna koraka: davanje instrukcija računaru i čitanje rezultata obrade. I dok se čitanje rezultata vrlo brzo učinilo koliko-toliko snošljivim uvođenjem štampača na kojima su se rezultati ispisivali, unošenje instrukcija - programiranje - se sastojalo od mukotrpnog unosa niza nula i jedinica. Ti nizovi su davali računaru upute kao što su: "saberi dva broja", "premjesti podatak s neke memorijske lokacije na drugu", "skoči na neku instrukciju izvan normalnog slijeda instrukcija" i slično. Programski jezici koji su omogućavali ovu komunikaciju nazivaju se ***mašinski programski jezici***. Kako je takve programe bilo vrlo složeno pisati, a još složenije čitati i ispravljati, ubrzo su se pojavili prvi programerski alati nazvani asembleri (engl. *assemblers*).

U ***asemblerskom*** jeziku svaka mašinska instrukcija predstavljena je mnemonikom koji je razumljiv ljudima koji čitaju program. Tako se sabiranje npr. obavlja mnemonikom ADD, dok se premještanje podataka obavlja mnemonikom MOV. Time se postigla bolja čitljivost programa, no i dalje je bilo vrlo složeno pisati programe i ispravljati ih jer je bilo potrebno davati sve, pa i najmanje instrukcije računaru za svaku pojedinu operaciju. Javlja se problem koji će kasnije, nakon niza godina, dovesti i do pojave C++ programskog jezika: potrebno je razviti programerski alat koji ce osloboditi programera rutinskih poslova te mu dopustiti da se usredotoči na problem koji rješava. Zbog toga se pojavljuje niz ***viših programska jezika***, koji preuzimaju na sebe neke "dosadne" programerske poslove. Tako je FORTRAN bio posebno pogodan za matematičke proračune, zatim BASIC koji se vrlo brzo učio, te COBOL koji je bio u pravilu namijenjen upravljanju bazama podataka.

Oko 1972. se pojavljuje jezik C, koji je direktna preteča današnjeg jezika C++. To je bio prvi jezik opšte namjene te je postigao neviđen uspjeh. Više je razloga tome: jezik je bio jednostavan za učenje, omogućavao je modularno pisanje programa, sadržavao je samo naredbe koje se mogu jednostavno prevesti u mašinski jezik, davao je brzi kôd. Jezik je omogućavao vrlo dobru kontrolu mašinskih resursa te je na taj način omogućio programerima da optimiziraju svoj kôd. Do unatrag nekoliko godina, 99% komercijalnih programa bili su pisani u C-u, ponegdje dopunjeni odsječcima u mašinskom jeziku kako bi se kritični dijelovi učinili dovoljno brzima.

No kako je razvoj programske podrške napredovao, stvari su se i na podrucju programskih jezika počele mijenjati. Složeni projekti od nekoliko stotina hiljada, pa i više redova više nisu rijetkost, pa je zbog toga bilo potrebno uvesti dodatne mehanizme kojima bi se takvi programi učinili jednostavnijima za izradu i održavanje, te kojima bi se omogućilo da se jednom napisani kôd iskoristi u više različitih projekata.

3.2. PROCEDURALNO PROGRAMIRANJE

Proceduralno programiranje se zasniva na posmatranju programa kao niza jednostavnih programskih cjelina: procedura. Svaka procedura je konstruisana tako da obavlja jedan manji zadatak, a cijeli se program sastoji od niza procedura koje međusobno sudjeluju u rješavanju zadatka.

Princip kojim bismo mogli obilježiti proceduralno strukturirani model jest *podijeli-pa-vladaj*: cjelokupni program je presložen da bi ga se moglo razumjeti pa se zbog toga on rastavlja na niz manjih zadataka – procedura, koje su dovoljno jednostavne da bi se mogle izraziti pomoću naredbi programskog jezika. Pri tome, pojedina procedura također ne mora biti riješena monolitno: ona može svoj posao obaviti kombinirajući rad niza drugih procedura.

Na primjer, pretpostavimo da treba napisati program za nalaženje korijena kvadratne jednačine. Ovaj problem mogli bismo riješiti u slijedećim koracima:

1. Unesi vrijednosti a,b,c
2. Izračunaj diskriminantu
3. U ovisnosti o vrijednosti diskriminante, izračunaj korijene
4. Odštampaj korijene

Ovakav programski pristup je bio vrlo uspješan do kasnih osamdesetih, kada su njegovi nedostaci postajali sve očitiji. Pokazalo se složenim istodobno razmišljati o problemu i odmah strukturirati rješenje. Umjesto rješavanja problema, programeri su mnogo vremena provodili pronalazeći načine da programe usklade sa zadanom strukturom. Također, današnji programi se pokreću pomoću miša, prozora, menija i dijaloga. Programiranje je *vođeno događajima* (engl. *event-driven*) za razliku od starog, sekvencijalnog načina.

Proceduralni programi su korisniku u pojedinom trenutku prikazivali niz ekrana nudeći mu pojedine opcije u određenom redoslijedu. No vođeno *događajima* znači da se program ne odvija po unaprijed određenom slijedu, već se programom upravlja pomoću niza događaja. Događaja ima raznih: pomicanje miša, pritisak na tipku, izbor stavke iz menija i slično. Sada su sve opcije dostupne istovremeno, a program postaje interaktivan, što znači da promptno odgovara na korisnikove zahtjeve i odmah (ovo ipak treba uvjetno shvatiti) prikazuje rezultat svoje akcije na monitoru računara. Kako bi se takvi zahtjevi jednostavnije proveli u praksi, razvijen je ***objektni pristup programiranju***.

Osnovna ideja je razbiti program u niz zatvorenih cjelina koje zatim međusobno sarađuju u rješavanju problema. Umjesto specijaliziranih procedura koje barataju podacima, radimo s objektima koji objedinjavaju i operacije i podatke. Pri tome je važno ***šta objekt radi***, a ***ne kako*** on to radi. Jedan objekt se može izbaciti i zamijeniti drugim, boljim, ako oba rade istu stvar.

3.3. PROGRAM

Sam računar, čak i kada se uključi u struju, nije kadar učiniti ništa korisno. Ono što mu nedostaje jeste pamet neophodna za koristan rad računara: programi. Pod programom podrazumijevamo niz naredbi u mašinskom jeziku koje procesor u računaru izvodi i shodno njima obraduje podatke, provodi matematičke proračune, ispisuje tekstove, iscrtava krivulje, itd.. Pokretanjem programa s diska, diskete ili CD-ROM-a, program se učitava u radnu memoriju računara i procesor počinje s mukotrpnim postupkom njegova izvođenja.

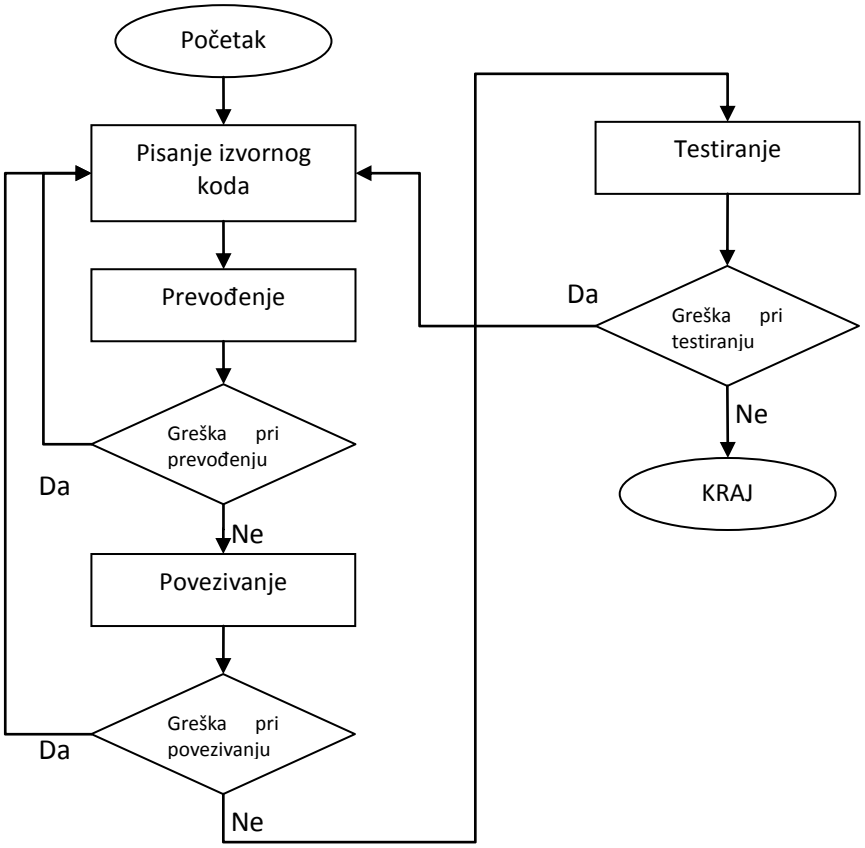
Programi koji se pokreću na računaru su u ***izvršnom obliku*** (engl. *executable*), razumljivom samo procesoru vašeg (i njemu sličnih) računara. U suštini se mašinski kôd sastoji od nizova binarnih cifara: nula i jedinica. Budući da su današnji programi tipično dužine nekoliko megabajta, naslućujete da ih autori nisu pisali izravno u mašinskom kôdu.

Gotovo svi današnji programi se pišu u nekom od ***viših programskih jezika*** (FORTRAN, BASIC, Pascal, C) koji su donekle razumljivi i ljudima. Naredbe u tim jezicima se sastoje od mnemonika.

Kombinovanjem tih naredbi programer slaže ***izvorni kôd*** (engl. *source code*) programa, koji se pomoću posebnih programa ***kompajlera*** (engl. *compiler*) i ***povezivača*** (engl. *linker)* prevodi u izvršni kôd. Prema tome, pisanje programa u užem smislu podrazumijeva pisanje izvornog kôda. Međutim, kako pisanje kôda nije samo sebi svrhom, pod pisanjem programa u širem smislu podrazumijeva se i prevođenje, odnosno povezivanje programa u izvršni kôd. Stoga možemo govoriti o četiri faze izrade programa:

1. pisanje izvornog kôda
2. prevođenje izvornog kôda,
3. povezivanje u izvršni kôd te
4. testiranje programa.

Prva faza programa je pisanje izvornog kôda. U principu se izvorni kôd može pisati u bilo kojem programu za uređivanje teksta (engl. *text editor*), međutim velika većina današnjih kompajlera i povezivača se isporučuje kao cjelina zajedno s ugrađenim programom za upis i ispravljanje izvornog kôda. Te programske cjeline poznatije su pod nazivom *integrisane razvojne okoline* (engl. *integrated development environment, IDE*). Nakon što je pisanje izvornog kôda završeno, on se pohrani u datoteku izvornog kôda na disku. Toj datoteci se obično daje nekakvo smisleno ime, pri čemu se ono za kôdove pisane u programskom jeziku C++ obično proširuje nastavkom .cpp, .cp ili samo .c, na primjer *neki_program.cpp*. Nastavak je potreban samo zato da bismo izvorni kôd kasnije mogli lakše pronaći.



Slika 25. Faze izrade programa

Slijedi prevođenje izvornog kôda. U integrisanim razvojnim okolinama program za prevođenje se pokreće pritiskom na neku tipku na ekranu, pritiskom odgovarajuće tipke na tastaturi ili iz nekog od *menija* (engl. *menu*) - ako kompajler nije integrisan, poziv je nešto složeniji. Kompajler tokom prevođenja *provjerava sintaksu napisanog izvornog kôda i u slučaju uočenih ili naslućenih grešaka ispisuje odgovarajuće poruke o greškama*, odnosno upozorenja. Greške koje prijavi kompajler nazivaju se *greškama pri prevođenju* (engl. c*ompile-time errors*). Nakon toga programer će pokušati ispraviti sve navedene greške i ponovo prevesti izvorni kôd - sve dok prevođenje kôda ne bude uspješno okončano, neće se moći pristupiti povezivanju kôda. Prevođenjem izvornog dobiva se datoteka ***objektnog kôda*** (engl. *object code*), koja se lako može prepoznati po tome što obično ima nastavak .o ili .obj (u našem primjeru bi to bio *neki_program.obj*).

Nakon što su ispravljene sve greške uočene prilikom prevođenja i kôd ispravno preveden, pristupa se ***povezivanju objektnih kôdova u izvršni kod***. U većini slučajeva objektni kôd dobiven prevođenjem programerovog izvornog kôda treba povezati s postojećim ***bibliotekama*** (engl. *libraries*). Biblioteke su datoteke u kojima se nalaze već prevedene gotove funkcije ili podaci. One se isporučuju zajedno s kompajlerom, mogu se zasebno kupiti ili ih programer može tokom rada sam razvijati. Bibliotekama se izbjegava ponovno pisanje vrlo često korištenih operacija. Tipičan primjer za to je biblioteka matematičkih funkcija koja se redovno isporučuje uz kompajlere, a u kojoj su definisane sve funkcije poput trigonometrijskih, hiperbolnih, eksponencijalnih i sl.. Prilikom povezivanja provjerava se mogu li se svi pozivi kôdova realizovati u izvršnom kôdu. Uoči li povezivač neku nepravilnost tokom povezivanja, ispisat ce poruku o grešci i onemogućiti generisanje izvornog kôda.

Ove greške nazivaju se ***greškama pri povezivanju*** (engl. *link-time errors*) - sada programer mora prionuti ispravljanju grešaka koje su nastale pri povezivanju. Nakon što se isprave sve greške, kôd treba ponovno prevesti i povezati. Uspješnim povezivanjem dobiva se ***izvršni kôd***. Međutim, takav izvršni kôd još uvijek ne garantuje da će program raditi ono što ste zamislili. Na primjer, može se dogoditi da program radi pravilno za neke podatke, ali da se za druge podatke ponaša nepredvidivo. U tom se slučaju radi o ***greškama pri izvođenju*** (engl. *run-time errors*). Da bi program bio potpuno korektan, programer treba testirati program da bi uočio i ispravio te greške, što znači ponavljanje cijelog postupka u lancu –

“ispravljanje izvornog kôda-prevođenje-povezivanje-testiranje'. Za ispravljanje grešaka pri izvođenju, programeru na raspolaganju stoje ***programi za otkrivanje grešaka*** (engl. *debugger*). Radi se o programima koji omogućavaju prekid izvođenja izvedbenog kôda programa koji testiramo na unaprijed zadanim naredbama, izvođenje programa naredbu po naredbu, ispis i promjene trenutnih vrijednosti pojedinih podataka u programu. Najjednostavniji programi za otkrivanje grešaka ispisuju izvršni kôd u obliku mašinskih naredbi. Međutim, većina današnjih naprednih programa za otkrivanje grešaka su *simbolički* (engl. s*ymbolic debugger*) - iako se izvodi prevedeni, mašinski kôd, izvođenje programa se prati preko izvornog kôda pisanog u višem programskom jeziku. To omogućava vrlo lagano lociranje i ispravljanje grešaka u programu. Osim grešaka, kompajler i linker redovno javljaju i ***upozorenja***. Ona ne onemogućavaju nastavak prevođenja, odnosno povezivanja kôda, ali predstavljaju potencijalnu opasnost. Upozorenja se mogu podijeliti u dvije grupe. Prvu grupu čine upozorenja koja javljaju da kôd nije potpuno korektan.

Kompajler ili povezivač će zanemariti našu pogrešku i prema svom nahođenju generisati kôd. Drugu grupu čine poruke koje upozoravaju da "nisu sigurni je li ono što smo napisali upravo ono što smo željeli napisati", tj. radi se o dobronamjernim upozorenjima na zamke koje mogu proizići iz načina na koji smo program napisali. Iako ce, unatoč upozorenjima, program biti preveden i povezan (možda čak i korektno), pedantan programer neće ta upozorenja nikada zanemariti - ona često upućuju na uzrok grešaka pri izvođenju gotovog programa. Za precizno tumačenje poruka o greškama i upozorenja neophodna je dokumentacija koja se isporučuje uz kompajler i povezivač.

3.4. C++ PROGRAMIRANJE

3.4.1. STRUKTURA C++ PROGRAMA

C++ je objektno orjentirani programski jezik kojim su pisani mnogi današnji programi koje srećete u svakodnevnom radu na računaru.
C/C++ program se počinje izvršavati iz funkcije main. Uzmimo za primjer sljedeći programski kod:

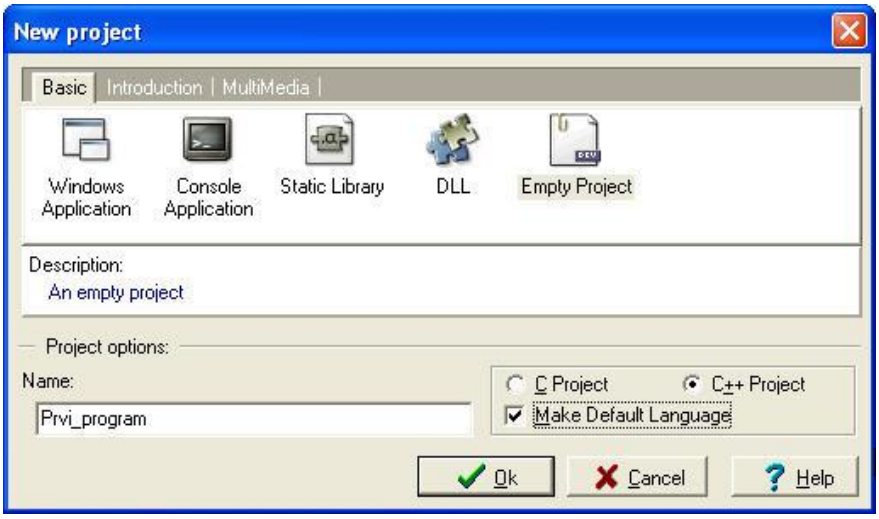
```
//Ovo je moj prvi program
#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World!";
system("pause");
return 0;
}
```

C program se sastoji iz blokova. Blok započinje znakom *{* a završava sa *}*.
Unutar funkcije main se mogu deklarisati varijable, pisati izrazi, pozivati druge funkcije, pisati komentari itd..

3.4.2. PRVI PROJEKAT

U ovom kursu za testiranje programskih kodova i njihovo kompajliranje je korišten besplatni IDE ***Dev C++*** koji se može preuzeti sa *http://www.bloodshed.net/devcpp.html*

Kada ste instalirali Dev-C++ pokrenite ga i idite na *File -> New -> Project*. Odaberite *''Empty Project''* te ga imenujte kao *''Prvi_program''* (bez razmaka). Označite dolje desno *C++ Project* i uključite kvačicu na *''Make Default Language''*. Kliknite na *OK*, zatim na *Save*.



Kada ste to napravili, idite *na File -> New -> Source File* i kliknite na *Yes*. S time ste dobili file gdje ćete pisati vaš C++ kod.
Pa napravimo jednostavan ''Hello World'' program za probu.

Upišite ovaj kod:

```
#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World!";
system("pause");
return 0;
}
```

Primjetite da svaka naredba u C++ mora završavati sa sa znakom ;
Sada taj kod treba kompajlirati. To ćete učiniti tako da pritisnete tipku F9 na tipkovnici. Program će se kompajlirati i pokrenuti.

Pa da objasnimo sada dijelove cijelog koda ovog jednostavnog programa.

#include <iostream> - Program zahtjeva od prevoditelja da u program uključi biblioteku *iostream* koja je standardna ulazno/izlazna biblioteka koja nam omogućuje ispis na ekranu.

NAPOMENA: ***#include*** nije naredba u C++ nego se radi o pretprocesorskoj naredbi *using namespace std;*

Svi elementi standardne C++ biblioteke su deklarirani u ovome što piše *namespace* sa imenom std.

int main () - Svaki program mora imati ni manje ni više nego jednu main funkciju. Sav kod unutar main zagrada se izvršava. Int predstavlja Integer (cijeli broj) što govori da će program pri završetku izvođenja programa vratiti cijeli broj.

cout << "Hello World"; - Ovo ispisuje Hello World! na ekran. Cout predstavlja standardni ispisni tok. Mogli ste taj kod napisati i ovako

```
cout << ''Hello World“ <<endl;
```

A možete i jednostavno nastaviti rečenicu u novi red tako da napišete

```
cout << "Hello World!" <<endl <<"Ja sam programer";
```

endl predstavlja End Line (kraj linije) odnosno ispis u novi red. Tako bi svaka rečenica koju napišete bila u redu ispod.
return 0; - Tom naredbom glavni program javlja operacijskom sustavu da je program uspješno završen.

3.4.3. KOMENTARI

Kod je potrebno ponekad komentirati kako se ne bi izgubili u kodu, ili jednostavno kao podsjetnik na nečega.

Ovako možete komentirati kod bez da taj tekst utječe na izvršavanje programa.

```
/* Ovo je moj prvi program, sa ovim načinom komentiranja
mogu svoje komentare pisati u više redova, tako da tu može
biti svega... */
```

```
#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World!"; // komentiram samo jednu liniju
char a;
cin >> a;
return 0;
}
```

3.4.4. VARIJABLE

U prošlom tutorijalu smo napisali kako ispisati rečenicu na ekran. U ovom tutorijalu ćemo govoriti o varijablama.

Što su varijable?

Varijable postoje u svakom programskom jeziku. U varijablama pohranjujete nekakve vrijednosti, brojeve, slova, i sve druge znakove. Evo primjera kako rečenicu iz prošlog tutorijala upisati u varijablu i zatim pomoć varijable ispisati istu na ekran.

```
#include<iostream>
using namespace std;
main()
{
string recenica;
recenica = "Hello World!";
cout << recenica;
char a;
cin >> a;
return 0;
}
```

Dakle sa ***string recenica*** deklariramo da će varijabla *''recenica''* biti string odnosno skup nekakvih znakova. String je tip podatka. O tome više u idućem tutorijalu. Zapamtite samo da se svaka varijabla mora deklarirati prije pridruživanja vrijednosti.

recenica = ''Hello World'';

Sa ovime varijabli ''recenica'' pridružujemo vrijednost ''Hello World!''

cout << recenica;

Sa ovim ispisujemo varijablu. Primjetite da nema navodnika. Ako želite nešto pored toga napisati što nije sadržano u varijabli možete to napisati ovako.

```
cout << recenica << " What's up?";
```

Dakle prvo ide varijabla *''recenica''* bez navodnika, zatim opet stavljamo strelice za ispis i u navodnike pišemo nastavak rečenice. Primijetite razmak između navodnika i slova W u drugoj rečenici. To smo napravili tako da riječi ne budu spojene jer će se ispisati u isti red.
Prisjetite se da ako želite u novi red ispisati nešto možete to napraviti sa ''<<endl''. Također u novi red možete ići ako napišete negdje u navodnicima **\n**, npr.

```
cout << recenica << "\n What's up?";
```

Idemo sada napraviti mali program koji će izračunavati dva broja.

```
#include<iostream>
using namespace std;

main()
{
int a, b, rezultat;
a = 7;
b = 12;
rezultat = a + b;
cout << "Zbroj ta dva broja je " << rezultat;

char x;
cin >> x;
return 0;
}
```

Kao što string označava niz znakova, tako int označava *integer* odnosno cijeli broj. Znači sve što će biti pridodano varijablama a, b i rezultat će biti cijeli broj (nikakvi decimalni brojevi, slova, drugi znakovi nego samo cijeli broj).
Primijetite da smo ovdje protiv nestajanja prozora koristili drugu varijablu da ne bi došlo do konflikta.

Deklarirati varijable možete ovako kao u primjeru a možete i svaku posebno na ovaj način:

```
int a;
int b;
int rezultat;
```

Nakon deklariranja varijabli slijedi naravno pridodavanje vrijednosti tim varijablama. Pridodajemo varijabli a broj 7, varijabli b broj 12 a varijabla rezultat će pohraniti onaj zbroj koji daju varijable a i b. U ovom slučaju 7 + 12 = 19. Znači varijabli "rezultat" se pridružuje vrijednost 19.

Varijable možete također deklarirati i pridružiti im vrijednost odjednom. Dakle ovako:

```
int a = 7;
int b = 12;
int rezultat = a + b;
```

Probajte sada napisati ovakav program

```
#include<iostream>
using namespace std;

main()
{
int a, b, rezultat;
cout << "Unesite prvi broj: ";
cin >> a;
cout << "Unesite drugi broj: ";
cin >> b;
```

```
rezultat = a + b;
cout << "Zbroj ta dva broja je " << rezultat;

char x
cin >> x;
return 0;
}
```

Dakle, ovo je skoro isto kao i prethodni primjer samo što ovdje imamo

```
cin >> a;
```

Program očekuje da ćete napisati neki broj i kada vi napišete taj broj on će ga spremiti u varijablu "a". Isto tako i dvije linije poslije za varijablu "b".

Signed i Unsigned
Ovo koristimo za varijable sa predznakom i bez predznaka. Ako stavite

unsigned int a;

onda varijabli "a" nećete moći pridružiti broj sa predznakom (npr. -12) a ako umjesto unsigned napišete signed ili ostavite bez toga onda ćete moći upisati predznak.

3.4.5. DOSEG VARIJABLI

Varijable mogu biti globalne ili lokalne. Globalne varijable su one varijable koje su deklarirane u glavnom tijelu *source* koda, izvan svih funkcija dok su lokalne varijable one varijable koje su deklarirane u funkciji ili bloku.

Primjer lokalnih varijabla

```
#include<iostream>
using namespace std;

main()
{
int a, b, rezultat;
a = 2;
b = 5;
rezultat = a + b;
cout << rezultat;

char x;
cin >> x;
return 0;
}
```

Primjer globalnih varijabli

```
#include<iostream>
using namespace std;

int a, b, rezultat;

main()
{ a = 2;
b = 5;
rezultat = a + b;
cout << rezultat;

char x;
cin >> x;
return 0;
}
```

Globalnim varijablama se može pristupiti bilo gdje u kodu, čak i u funkcijama dok je lokalnim varijablama moguće pristupiti samo unutar vitičastih zagrada { i }

NAPOMENA: sve varijable **moraju** početi sa slovom ili donjom crticom **_** . Ne smiju početi brojevima niti ičim drugim. Osim toga, sve varijable u svom nazivu smiju imati samo slova, brojeve (od drugog mjesta na dalje), i crtice (**_**), a ne smiju sadržavati razmake, navodnike i slične simbole.

Također bitno je to da vam se varijable ne smiju zvati sljedećim imenima:

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq

Jer su to ključne riječi i operatori jezika C++!!!!

Još jedna napomena: C++ je ***case sensitive*** jezik, odnosno ***osjetljiv na velika i mala slova.*** Tako da recenica i RecEnica nije ista varijabla.

Kao što smo već rekli, u sljedećem tutorijalu ćemo se pozabaviti tipovima podataka (to vam je ono int, char, string i ostali...)

3.4.6. TIPOVI PODATAKA

Ovaj tutorijal se nadovezuje na varijable, jer kako smo u prošlom tutorijalu napisali da *int*, *string* i *char* označavaju tip podatka koji će biti pridružen varijabli. Pa ćemo sada detaljnije objasniti te i ostale tipove podataka.

Kada programiramo, spremamo varijable u memoriju računara, ali računar mora znati što ćemo (kakav tip podatka) spremiti u varijablu. Jedan jednostavan broj neće zauzeti istu količinu memorije kao jedan veliki tekst ili veliki broj, i neće biti interpretirano na isti način.

Memorija računara je organizirana po bajtovima (engl. *bytes*). Jedan bajt je minimalna količina memorije koju možemo sačuvati u C++. Jedan bajt može pohraniti relativno mali broj podataka. Jedno slovo ili jedan cijeli broj od 0 do 255.
Osim toga, C++ može upravljati mnogo kompleksnijim tipovima podataka koje dolaze grupiranjem bajtova. Kao što su dugački brojevi (*long numbers*) i slično.
Evo jedne tablice sa osnovnim tj. bitnim tipovima podataka.

Ime	Opis	Veličina	Domet
char	Character - jedan znak ili mali broj	1 bajt	S predznakom od -128 do 127 Bez predznaka od 0 do 255
short int (short)	Short integer - kratki cijeli broj	2 bajta	S predznakom: od - 32768 do 32767 Bez predznaka: od 0 do 65535
int	Integer - cijeli broj	4 bajta	S predznakom: - 2147483648 do 2147483647 Bez predznaka: od 0 do 4294967295
long int (long)	Long integer - dugački cijeli broj	4 bajta	S predznakom: - 2147483648 do 2147483647 Bez predznaka: od 0 do 4294967295
bool	Bolean - istina ili neistina	1 bajt	Istina ili neistina
float	Floating point number	4 bajta	3.4e +/- 38 (7 brojeva)
double	Double precision floating point number	8 bajtova	1.7e +/- 308 (15 brojeva)
long double	Long double precision floating point number	8 bajtova	1.7e +/- 308 (15 brojeva)
wchar_t	Wide character - "divlji" znak	1 bajt	Npr. japansko slovo

3.4.7. KONSTANTE (NEPROMJENJIVE)

Varijable su promjenjive dok su konstante su izrazi sa fiksnim, nepromjenjivim vrijednostima.

Morate inicirati konstantu kada ju napravite, i ne možete joj pridružiti novu vrijednost kasnije. Poslije kada je konstanta inicirana njezina vrijednost je nepromjenjiva.

3.4.8. LITERAL CONSTANTS (KONKRETNE KONSTANTE)

C++ ima dva tipa konstanti: konkretne i simbolične (literal i symbolic).

Literal konstanta je vrijednost upisana direktno u program kada god je to potrebno. Npr.

int godine = 24;

godine je varijabla tipa integer, a 24 je literal konstanta. Ne možete pridružiti vrijednost na 24, i ta vrijednost ne može biti promijenjena.

***Symbolic constants* (simbolične konstante)**

Simbolična konstanta je konstanta koja je predstavljena po imenu, baš kao i varijabla. Ali nakon što je konstanta inicirana, njezina vrijednost ne može biti promjenjena. Ako imate jednu integer varijablu imenovanu "studenti" i drugu imenovanu "ucionica", možete procijeniti koliko studenata imate, i broj učionica, te ako znate da je 15 studenata po razredu.

```
sudenti = ucionice * 15;
```

U ovom primjeru, broj 15 je Literal konstanta. Kod bi bio puno jednostavniji za čitanje i jednostavniji za održavati ako biste tu vrijednost zamijenili simboličnom konstantom.

```
studenti = ucionice * ucenika_po_ucionici;
```

Ako kasnije odlučite promijeniti broj studenata po ucionici, mozete to učiniti tako da definirate konstantu ucenika_po_ucionici bez potrebe za izmjenama u svakom dijelu koda gdje ste koristili tu vrijednost.

Definiranje konstanti sa #define

Da biste definirali konstantu na sa staromodnim, i lošim načinom unijet ćete

```
#define ucenika_po_razredu 15;
```

Primjetite da ucenika_po_ucionici nema konkretni tip (int, char itd..). Svaki puta kad preprocesor vidi riječ ucenika_po_razredu upisat će broj 15 u tekst.

Zbog toga što se preprocesor pokreće prije kompajlera, računar nikad ne vidi vašu konstantu, on vidi broj 15.

Definiranje konstanti sa const

Iako #define radi, postoji bolji, ukusniji način definiranja konstanti u C++

```
const unsigned int ucenika_por_ucionici = 15;
```

U ovom primjeru se isto deklarira simbolična konstanta imenovana ucenika_po_ucionici, ali ovaj puta ucenika_po_ucionici je napisano kao unsigned int (bez predznaka, cijeli broj)

S ovime imate više za tipkati ali nudi nekoliko prednosti. Najveća razlika je u tome što ova konstanta ima tip i kompajler može prisiliti da to bude korišteno po svom tipu.

3.4.9. ARITMETIČKI OPERATORI

Operator je simbol koji predstavlja specifičnu akciju. Već smo kod primjera zbrajanja koristili operator "+". Osim operatora + imamo još nekoliko aritmetičkih operatora.

Evo tablice za pregled operatora.

Operator	Namjena	Primjer	Rezultat
+	Zbrajanje	5 + 6	11
-	Oduzimanje	7-3	4
*	Množenje	4 * 4	16
/	Dijelenje	12 / 6	2
%	Dijeljenje (s ostatkom)	5 % 2	1

% operator se naziva još i *modulus operator*.

Aritmetički operatori jednako dobro rade sa negativnim brojevima kao i sa pozitivnim, sa iznimkom modulus operatora, rade sa cijelim brojevima jednako dobro kao i sa brojevima sa pomičnim zarezom.

Neki programski jezici imaju eksponent operator, što nije slučaj kod C++. Umjesto toga C++ ima ugrađenu funkciju *pow* koja je definirana u standardnoj biblioteci *cmath*
pow funkcija ima dva argumenta. Prvi argument je baza (glavni broj), a drugi broj je eksponent.
Pogledajmo primjer:

```
#include <iostream>
#include <cmath> // ukljucili smo biblioteku cmath
using namespace std;

int main()
{
double baza, eksponent, rezultat;
cout << "Unesite broj ";
cin >> baza;
cout << "Unesite eksponent ";
cin >> eksponent;
rezultat = pow(baza, eksponent);
cout << "Rezultat = " << rezultat;

char x;
cin >> x;
return 0;
}
```

Prvo novo što morate primjetiti je to da smo uključili biblioteku cmath. Zatim smo deklarirali broj, eksponent i rezultat kao double tip. (Zbog ogromnih brojeva ako npr. upišete 10 na 10-u). Zatim smo im sa cin >> pridružili vrijednosti, te su te vrijednosti zapravo parametri u funkciji *pow* .

3.4.10. ODNOSNI OPERATORI

U programskom jeziku C++ kao i u mnogim drugim programskim jezicima postoje odnosni operatori. Takvi operatori se npr. koriste ako želite napraviti program koji će iz baze podataka odrediti osobe sa više od 30 godina i slično.

Evo tih odnosnih operatora:

Operator	Značenje
>	Više od...
<	Manje od...
>=	Više ili jednako
<=	Manje ili jednako
==	Jednako
!=	Različito (nejednako)

3.4.11. ODNOSNE NAREDBE

Kao i aritmetički operatori, odnosni operatori su binarni tj. uspoređuju dva operanda. Naredba sa dva operanda i odnosnim operatorom zove se odnosna naredba (eng. *relational expression*.).
Rezultat odnosne naredbe je Boolean vrijednost odnosno istinu ili laž (*true* ili *false*). S ovom tablicom možete vidjeti kako to funkcionira

Odnosna naredba	Vraćena vrijednost
4 == 4	Istina
4 < 4	Laž
4 <= 4	Istina
4 > 4	Laž
4 != 4	Laž
4 == 5	Laž
4 < 5	Istina
4 >= 5	Laž
4 != 5	Istina

U ovoj tablici se koriste konkretne (*literal*) vrijednosti koje ne mogu biti promijenjene. 4 je konkretna vrijednost (konstanta), i ona se ne može mijenjati (moglo bi se mijenjati da je umjesto konstanta koristimo varijable)

Isprobajmo sada ovaj kod koji umjesto konstanti koristi varijable

```
#include <iostream>
using namespace std;
int main()
{
int a = 4, b = 5;
cout << (a > b) << endl;
cout << (a >= b) << endl;
cout << (a == b) << endl;
cout << (a <= b) << endl;
cout << (a < b) << endl;

char x;
cin >> x;
return 0;
}
```

Ovaj program će ispisati:
0
0
0
1
1

S time da 0 predstavlja laž (*false*) a 1 istinu (*true*).

3.4.12. NAREDBA IF

Naredbu IF koristimo onda kada želimo izvršiti neki kod samo ako je vrijednost nekog odnosnog izraza istinita. Evo primjera:

```
#include<iostream>
using namespace std;

main()
{
string password;
cout << "Unesite password: ";
cin >> password;

if(password=="G2105Z") {
cout << "Password tocan!"; }

char x;
cin >> x;
return 0;
}
```

Definirali smo string password, sa cin naredbom upisali ono što korisnik upiše u varijablu password, zatim slijedi provjera sa IF naredbom.

Sintaksa IF naredbe je zapravo:
if(uvjet) { kod koji se izvršava }

Naravno nije IF naredba ograničena samo na provjeravanje točnosti. Možete koristiti sve odnosne operatore koje smo objasnili u prošlom tutorijalu. Na primjer:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine < 18 ) {
cout << "Maloljetni ste!"; }

char x;
cin >> x;
return 0;
}
```

U svrhu učenja isprobajte sve odnosne operatore. Npr. probajte napisati mali program koji će provjeriti da li je broj djeljiv sa 2 (bez ostatka).

U sljedećem tutorijalu ćemo učiti o IF...ELSE naredbi koja vam pruža mogućnost da uz IF naredbu imate i naredbu koja će izvršiti neki kod ukoliko uvjet nije zadovoljen.

3.4.13. IF...ELSE NAREDBA

If...Else naredba je proširenje If naredbe. Kao što smo već rekli u prošlim tutorijalu, If naredba omogućuje izvršavanje nekog koda ukoliko je uvjet zadovoljen, a ukoliko nije neće se izvršiti ništa.

Ako bi smo htjeli napraviti da ukoliko uvjet nije zadovoljen da se izvrši neki drugi kod onda koristimo If...Else naredbu.

Evo primjera kako koristiti If...Else naredbu

```
#include<iostream>
using namespace std;

main()
{
string password;
cout << "Unesite password: ";
cin >> password;

if(password=="G2105Z") {
cout << "Password tocan!"; }
else {
cout << "Password netcan!"; }

char x;
cin >> x;
return 0;
}
```

Kao sto vidite kod je isti kao i za if naredbe, else pa vitičaste zagrade i unutra kod za izvršavanje.

Evo i drugog primjera:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine < 18 ) {
cout << "Maloljetni ste!"; }
else {
cout << "Punoljetni ste!" }

char x;
cin >> x;
return 0;
}
```

3.4.14. IF...ELSE IF...ELSE NAREDBA

U prošlom tutorijalu smo govorili o IF...Else naredbi. Ovdje ćemo proširiti tu naredbu tako da možemo ispitati više uvjeta. Za to koristimo If...Else-If...Else naredbu.

Evo primjera:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine <= 17 ) {
cout << "Maloljetni ste!"; }

else if (godine>=18 and godine<=39) {
cout << "Punoljetni ste"; }

else if (godine>39 and godine<70) {
cout << "Najljepše godine"; }

else {
cout << "Stari ste"; }

char x;
cin >> x;
return 0;
}
```

Dakle sa ovom naredbom možemo ispitati više uvjeta. U ovom primjeru dosta koristimo odnosne operatore, pa ako niste naučite ih (4 tutorijala ispred ovoga).

Sve je isto kao i naredba if...else samo što ovdje možete više puta staviti naredbu *else if* kako biste ispitali više uvjeta.

else if možete postaviti koliko god hoćete puta, ali ako imate jako puno mogućih uvjeta onda vam preporučam da pogledate sljedeći tutorijal u kojem ćemo vam objasniti kako se koristi *switch* naredba za ispitivanje mogućih uvjeta.

3.4.15. NAREDBA WHILE

Oba petlja nam sluzi za definiranje ciklusa sa nepoznatim brojem ponavljanja.
Format naredbe je:

Sintaksa:

```
while (uvjet)
{
naredbe;
}
```

Na primjer:

```
broj=1;
while (broj<5)
{
broj=broj+1; // proizvoljna naredba
}
```

U ovom primjeru broj je manji od 5 i ova petlja će se izvršiti, tako da će se povećati za 1. Slijedeći put će broj biti jednako 2 pa će se petlja opet izvršiti. Da bi se onda kada broj dobije vrijednost 5, petlja preskočila, tj. ne bi se izvršila i izvršio bi se ostatak koda, jer se nije ispunio uvjet koji glasi da broj mora da bude manji od 5.

```
#include<iostream.h>
int main ()
{
int broj=1;
while (broj<5)
{
broj=broj+1; //ovo se može napisati i broj+=1;
}
cout<<"Ovo je broj: "<<broj<<endl;
return 0;
}
```

Rezultat ovog programa je:

Ovo je broj: 5

3.4.16. NAREDBA FOR

Naredba for

Kod while petlje smo imali da inicijalizaciju, uvjet i promjenu vrijednosti imamo na više različitih mjesta, dok je kod for petlje to sve sadržano u jednom redu koda tj. između malih zagrada.

Format naredbe:
for (inicijalizacija; uvjet; promjena vrijednosti) naredba;.
ili korištenjem blokova:
for (inicijalizacija; uvjet; promjena vrijednosti)
{
naredbe;
}

Uvjet mora biti logički izraz, dok inicijalizacija i promjena vrijednosti mogu biti bilo kakvi izrazi. Petlja će se izvršavati dok je uvjet tačan.

Na primjer:

```
//Brojač unatrag
#include <iostream.h>
int main ()
{
for (int i=10; i>0; i--) {
cout << i << ", ";
}
cout << "PALI!";
return 0;
}
```

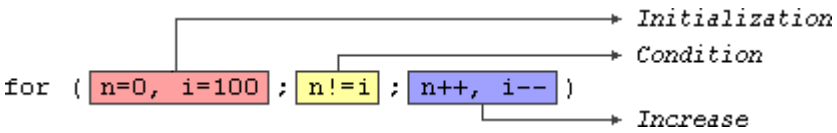
Rezultat ovog programa:
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, PALI!

Ukoliko dodamo na kraju ovog dijela koda "<<endl;"
cout << i << ", "<<endl;
Što će značiti da poslije svakog ispisa kursor šalje u novi red.

Nakon ovog rezultat ce biti:

10,
9,
8,
7,
6,
5,
4,
3,
2,
1,
PALI!

Evo na ovoj slici se jasno ilustrira format for petlje:



3.4.17. NAREDBA SWITCH

Ova naredba služi za provjeravanje više uvjeta od jednom i izvršavanja onoliko radnji koliko je uslova zadovoljeno odnosno tačno. Ova naredba je korisna ukoliko trebamo provjeriti više "stvari" a da ne petljamo sa if i else if petljom :).

Format ove naredbe je:

switch (varijabla){

case mogucnost1:

```
naredba;
break;

case mogucnost2:
naredba;
break;

...

}
```

Pa da svaki red koda objasnimo pojedinačno:

switch (varijabla){
Ovaj dio prijavljuje varijablu (promjenljivu) čiju ćemo tačnost ispitivati.

case mogucnost1:

Ovaj dio provjerava da li je varijabla iz prvog reda jednaka nizu znakova (u ovom slučaju) mogucnost1. Obavezno : (dvotačka) na kraju.

naredba;
Ovdje upisujemo naredbu ili više njih koje će se izvršavati ukoliko je zadovoljen uvjet.

break;
Ovom riječju se zatvara niz naredbi i daje se mogućnost novom uslovu.

case mogucnost2:
naredba;
break;

Ovo je samo primjer kako treba postaviti drugi uvjet, tako ćemo dodavati i ostale. Ukoliko želimo da postavimo neki defaultni uvjet tj. naredbe koje će se izvršiti ukoliko ni jedan uvjet nije zadovoljen onda ćemo dodati slijedeće:

default:
naredbe;

Ovdje na kraju ne ide *break*; zato što se podrazumjeva da će *default* kod biti na kraju tj. zadnji uvjet i da neće biti više uvjeta.

Ovo sve malo zvuči zamršeno ali na konkretnom primjeru će to drukčije (jednostavnije) izgledati.

```
#include<iostream.h>
int main()
{
int broj=0;
switch (broj){

//provjerava da li je broj=1
case 1:
```

```
//ako postoji ispisuje 'Broj je 1!!!'
cout<<"Broj je 1!!!"<<endl;
break;

case 2:
cout<<"Broj je 2!!!"<<endl;
break;

//Ukoliko broj nije jednak ni 1 niti 2 onda ispisuje
default:
cout<<"Broj je 0 (nula) !!!"<<endl;

}

return 0;
}
```

Evo sad za one koji misle da je lakše sa if petljama ispetljati više uvjeta:

```
switch (x) {
case 1:
cout << "x je 1";
break;
case 2:
cout << "x je 2";
break;
default:
cout << "x je nepoznato";
}


if (x == 1) {
cout << "x je 1";
}
else if (x == 2) {
cout << "x je 2";
}
else {
cout << "x je nepoznato";
}
```

Vidimo jednu if i jednu else if petlju, zamislite da moramo ispitati 50 vrijednosti tek onda bi bilo petljanja. Zato toplo preporučujem naredbu switch kod ispitivanja više različitih vrijednosti. Dok kod jednostavnih tu je naravno
if i if else.

3.4.18. JEDNODIMENZIONALNI NIZOVI

Dosadašnji primjeri koje smo radili mogli su da obrađuju samo jednu vrijednost, kombiniranje više vrijednosti jako komplicira rješenje.

Ako imamo 5 različitih varijabli:
a,b,c,d,e i vrijednost svake se unosi sa tipkovnice, to bi izgledalo ovako

```
cout<<"Unesi a: ";
cin>>a;
cout<<"Unesi b: ";
cin>>b;
cout<<"Unesi c: ";
cin>>c;
cout<<"Unesi d: ";
cin>>d;
cout<<"Unesi e: ";
cin>>e;
```

To predstavlja problem ako moramo unijeti 50 i više vrijednosti. Zato koristimo jednodimenzionalni nizovni tip. Pa će ovaj gore primjer izgledati ovako:

```
int niz[20],i;

for (i=0;i<=4;i++)
{
cout<<"Unesi vrijednost "<<i+1<<". varijable: ";
cin>>niz[i];
}
```

Pa da objasnimo što radi koji dio kôda:

int niz[20],i;

u ovom dijelu smo deklarirali varijablu niz i u srednje zagrade smo stavili koliko maksimalno mjesta može zauzeti u memoriji (u našem primjeru 20)

for (i=0;i<=4;i++)

Pomoću for petlje povećavamo vrijednost varijable "*i*" svaki put za jedan više. Povećavanje će ići sve dok vrijednost "*i*" ne dostigne 4. Zašto 4? Zato što smo htjeli da unosimo *5* varijabli pa ćemo imati 0,1,2,3,4 to je ukupno *5* brojeva. Mogli bismo staviti *i=1* pa bi onda postavili *i<=5* ali se u praksi pogotovo kod c++ početna vrijednost brojača postavlja na 0 dok je kod Pascala od 1 itd..

cout<<"Unesi vrijednost "<<i+1<<". varijable: ";

Kako se varijabla "*i*" bude povećavala tako će i ispis ovog koda biti drugačiji.
Npr: za *i=0* ispis će biti "Unesi vrijednost 1. varijable: ".
sad vidimo opet da piše 1., sigurno se pitate zašto nije 0.
Postavili smo i+1 zato što će na početku "i" biti 0 i plus ono jedan=1 i zato je 1. ;)

cin>>niz[i];

Jednostavna naredba kojom unosimo vrijednost za *i-ti* element niza (element sa indeksom *i*).

1. Napisati program kojim se računa zbir sljedećeg reda 2+4+6+…n (prirodan broj učitati na početku).
2. Napisati program kojim se računa zbir sljedećeg reda 1+1/3+1/5+1/7+…1/n (prirodan broj učitati na početku programa).
3. Profesor ispravlja n testova učenika koji pripadaju 3t1 i 3t2 razredu. Potrebno je napisati program koji će učitati ocjene n testova. Uz svaku ocjenu testa treba učitati i oznaku razreda kojem test pripada. Cilj programa je izračunati srednju ocjenu posebno za razred 3t1 a posebno za razred 3t2.
4. Napisati program koji će na ekranu ispisati sve cijele brojeve između D i G. Iznose cijelih brojeva D i G korisnik unosi na početku programa.
5. Učitati broj učenika jednog razreda i zatim težine svakog učenika-ce. Uz težinu treba učitati i pol (M ili Z). Izračunati srednju težinu učenika-ce posebno.
6. Učitati cijeli broj n iz intervala 1-80. U slučaju da n nije između 1 i 80, učitavanje treba ponavljati sve dok učitani broj ne bude u zadanim granicama. Program treba ispisati n zvjezdica u jednom redu.
7. Jedna bakterija podijeli se na dvije. U sljedećem ciklusu svaka od dvije nastale bakterije podijeli se ponovo itd.. Nakon koliko ciklusa će nastati više od 10000 bakterija?
8. Mate je u banku uložio ušteđevinu od 5500 eura. Ako je godišnja kamara 6,5% napisati program koji ce izračunati nakon koliko godina će imati ušteđevinu od 12000 eura. Ispisati izračunate godine i tačan iznos u eurima.
9. Napisati program koji ispisuje tablicu vrijednosti funkcija sinX i cosX za vrijednosti varijable X iz intervala od 0 do 5 u koracima po 0,25. Tablica mora imati ovakvo zaglavlje:
    i. X sin(x) cos(x)
10. Napisati program koji će omogućiti korisniku da upisuje cijele brojeve toliko dugo dok ne upiše broj 0. Nakon toga program treba ispisati srednju vrijednost unesenih brojeva. Broj 0 se ne računa.
11. Ispisati sve parne brojeve između 1 i 200. Izračunati srednju vrijednost tih brojeva.
12. Učitati dva prirodna broja i izvršite cjelobrojno dijeljenje uzastopnim oduzimanjem.
    Upustvo: Dijeljenje se može svesti na oduzimanje tako da od djeljenika uzastopce oduzimamo djelitelj sve dok ne dobijemo rezultat manji od djelitelja. Broj oduzimanja je traženi rezultat.
13. Napisati program za učitavanje dvodimenzionog niza maksimalne dimenzije 10x10. Odštampati učitanu matricu i inverznu matricu.
14. Napisati program za učitavanje dvodimenzionog niza maksimalne dimenzije 10x10. Odštampati elemente na glavnoj dijagonali i ispisati njihovu sumu.
15. Napisati program za učitavanje dvodimenzionog niza maksimalne dimenzije 10x10. Odštampati elemente na sporednoj dijagonali i ispisati njihovu sumu.
16. Napisati program za učitavanje dvodimenzionog niza maksimalne dimenzije 10x10. Odštampati pozitivne elemente niza, a zatim negativne. Ispisati koliko ima pozitivnih, odnosno negativnih brojeva u nizu