# Rad sa stilovima (kreiranje, primena, izmena)

 Zadaci koji treba da se realizuju:

1. Na ceo tekst primeni font Verdana velicine 11pt
2. Uključiti dokument Navigation Pane (tab View/Show)
3. Napravi tri stila Naslov1 nivoa 1, Naslov2 nivoa 2, Naslov3 nivoa 3 koji se baziraju redom na stilovima Heading 1, Heading 2, Normal.
	1. Naslov1 – sve velika slova size 18, bold, centriran
	2. Naslov2 – sve velika slova size 14, italic, poravnat po levoj strani
	3. Naslov3 - size 12, underline, poravnat po levoj strani i uvučen za 2cm
4. Primeniti stil Naslov1 na tekst crvene boje, Naslov2 na tekst zelene boje, a Naslov3 na tekst plave boje
5. Sve naslove centrirati. Obezbediti da Naslov1 uvek počinje na novoj stranici, a Naslov2 i Naslov3 da budu uvek na istoj stranici na kojoj je sledeći paragraf.
6. Napraviti stil Proba1 sa sledećim karakteristikama: Verdana 12, poravnat po obe strane prva linija uvučena za 1cm, prored 1,5 linija, praznina posle paragrafa 6pt.
7. Napraviti stil Proba2 sa sledećim karakteristikama: Verdana 10, poravnat po obe strane prva linija uvučena za 1cm, prored 1linija, praznina posle i pre paragrafa paragrafa 3pt.
8. Napraviti stil Proba3 sa sledećim karakteristikama: Verdana 9, poravnat desnoj strani, uvučen s desne strane 3cm, prored 12pt linija, praznina posle i pre paragrafa 3pt, uokvren sa desne i donje strane linijom plave boje.
9. Stilov Proba1(2,3) primeiti na prvi paragrafe posle teksta pisanog stilom Naslov1(2,3)
10. Modifikovati stil Proba3 tako da su slova italic
11. Formiraj stil Podvucen za formatiranje karaktera tako da su crvene boje podvučeni dvostrukom zutom linijom
12. Primeni stil Podvucen na sve reci algoritam u fajlu.
13. Formiraj stil Lista1 baziran na List, kojim definišeš listu sa dva nivoa nabrajanja prvi nivo a) b).. drugi nivo neki bullet
14. Stil Lista1 primeni na deset poslednjih paragrafa ovog dokumenta, tako da su italik paragrafi drugog nivoa

Uvod

Zbog toga je došlo do razvoja programskih jezika koji omogućavaju zapis niza instrukcija računaru koje se ne mogu višeznačno protumačiti. Jednoznačnost svake konstrukcije programskog jezika je njegova najbitnija karakteristika. Programski jezik je skup pravila kojim se predstavljaju instrukcije i opisuju podaci računaru.

Kratak pregled razvoja programskih jezika

Za komunikaciju između ljudi koriste se jezici. Jezik je sredstvo za predstavljanje i prenošenje informacija. Čovek mora na neki način da saopšti računaru niz instrukcija koje on treba da izvrši. Programski jezik je sredstvo kojim čovek saopštava računaru program. Korišćenjem programskog jezika komuniciraju čovek i računar. Prirodni jezici dopuštaju nejednoznačnost i nepreciznost. Računar može „razumeti“ samo formalni zapis, ne tolerišući ni najmanje nepreciznosti. Zbog toga je došlo do razvoja programskih jezika koji omogućavaju zapis niza instrukcija računaru koje se ne mogu višeznačno protumačiti. Jednoznačnost svake konstrukcije programskog jezika je njegova najbitnija karakteristika. Programski jezik je skup pravila kojim se predstavljaju instrukcije i opisuju podaci računaru.

Programske jezike možemo podeliti po stepenu zavisnosti programskog jezika i računara na:

mašinski jezik

simbolički jezik

jezici višeg nivoa

Mašinski i simbolički jezici su zavisni od računara i njih nazivamo mašinski zavisnim jezicima. Jezici višeg nivoa pripadaju skupini koje nazivamo mašinski nezavisni jezici.

Mašinski zavisni jezici

Najjednostavniji programski jezik je interni jezik računara – mašinski jezik. Najvažniji delovi računara izgrađeni su od elektronskih i magnetnih komponenti, koje karakterišu dva stabilna stanja (ima napona – nema napona, teče struja - ne teče struja, severni pol – južni pol). Zbog tehničke realizacija računara u računari se sve predstavlja pomoću dva simbola 0 i 1, binarnom azbukom. Jedan znak binarne azbuke predstavlja bit (binary digit). Sve informacije u računaru predstavljaju se nizom bitova grupisanim u grupe od 8, 16, 32, 64 bita. Grupa od 8 bitova naziva se bajt (byte).

Mašinski jezik je izgrađen nad binarnom azbukom. Sve operacije i svi podaci predstavljeni su nizom bitova. U zavisnosti od arhitekture računara ti nizovi bitova imaju različita tumačenja i različite su veličine.

Kako se mašinski program sastoji od niza nula i jedinica, i zahteva dobro poznavanje načina rada i arhitekture određenog računara, vrlo je teško programirati na njemu. Programi na prvim računarima bili su zapisani mašinskim jezikom, i to je uslovilo da uzak krug ljudi piše i održava programe.

 Programski jezici višeg nivoa

Da bi se premostile poteškoće u programiranju na asemblerskim jezicima veoma rano počinju da se razvijaju mašinski nezavisni jezici, drugim rečima jezici višeg nivoa. Korišćenjem jezika višeg nivoa opis naredbi i podataka vrši se na način blizak prirodnom (engleskom) jeziku. Ovim jezicima jednoj naredbi odgovara više naredbi simboličkog jezika. Važno je napomenuti da ovi jezici imaju visok stepen nezavisnosti u odnosu na arhitekturu računara i operativni sistem na kojem se izvršavaju.

S obzirom da računar razume samo program napisan na mašinskom jeziku svaki program pisan jezikom višeg nivoa mora se prevesti na mašinski jezik. Na osnovu načina prevođenja i izvršavanja jezike višeg nivoa delimo na kompajlerske i interpreterske jezike. Prvo su nastali kompajlerski jezici Algol, Fortran, Cobol, PL/I... Kod ovih jezika izgrađuju se programi za prevođenje (kompajleri) kojim se ceo program napisan na višem programskom jeziku prevodi u njemu ekvivalentan mašinski program, koji se može izvršavati na računaru. Kod interpreterskih jezika program na višem programskom jeziku se prevodi i izvršava instrukcija po instrukcija. Primeri interpreterskih jezika su Lisp, Prolog, Basic, ...

U početku se razlikovala primena računara u oblasti poslovanja od primene računara u nauci i tehnici. Prvu je karakterisao veliki broj ulazno/izlaznih podataka i relativno jednostavan opis obrade podataka, pa su i jezici namenjeni toj primeni zadovoljavali te karakteristike (Cobol,...). Primena u nauci i tehnici karakterisana je malim brojem ulazno/izlaznih podataka a zato veoma složenim opisom obrade, pa su razvijani jezici za te namene poput Fortrana, Algola,.... . U to vreme programski jezik PL/I bio je dovoljno dobar za primenu u poslovanju kao i za primenu u oblasti složenih numeričkih izračunavanja. Današnjim razvojem jezika gubi se ova podela i savremeni programski jezici mogu da se koriste ravnopravno u svim oblastima.

Prema načinu rešavanja problema možemo izvesti podelu viših programskih jezika na proceduralne i deklarativne programske jezike. U proceduralnim jezicima jezikom dajemo računaru kompletan skup instrukcija kojim se rešava problem, tj. dajemo mu algoritam za rešavanje zadataka. Ovim jezicima opisujemo **kako** se rešava dati problem. Manje više svi poznatiji viši programski jezici su ovog tipa: Fortran, Cobol, Basic, Pascal, C… kao i mašinski zavisni jezici. Kasnije u razvoju programskih jezika dolazi se do ideje da se problemi opisuju, a da sam interpreter ima ugrađene postupke kako da reši opisani problem i tako dolazi do razvoja deklarativnih jezika. Ovim jezicima opisujemo **šta** znamo o problemu i šta želimo da dobijemo rešavajući problem, a sistem (interpreter) sam dolazi do postupka za rešavanje problema. Primeri deklarativnih jezika su Prolog, SQL, ... Sistem ima ugrađen algoritam kako se dolazi do rešenja, postojanje i pisanje opšteg algoritma za nalaženje rešenja je glavna poteškoća u razvoju ovih jezika, pa su zato deklarativni jezici obično specijalizovani za određene vrste problema. Koliko god da je ova ideja napredna u praksi je vrlo teško u potpunosti ispuniti deklarativnost.

Neke deklarativne jezike možemo podeliti na relacijske i funkcijske. Osnovni objekti u prvima su relacije, a u drugima su funkcije. Ideja razvoja funkcijskih jezika je u spajanju unapred definisanih funkcija, koje imaju svoje ulaze i izlaze, u cilju dobijanja rešenja problema. Programer koji koristi takav programski jezik gleda na razvoj programa kao na način spajanja elementarnih funkcija, čime se dobija sistem koji izračunava željeni rezultat. Primeri funkcijskih jezika su LISP, LOGO, ML, ... .

Posebnu klasu čine objektno orijentisani jezici, kod kojih su prisutni i proceduralni i neproceduralni elementi. U proceduralnim jezicima isticani su postupci, način realizacije (procedura) neke više ili manje složene akcije. Podaci su na neki način u potcenjenom položaju u odnosu na algoritam. Moramo priznati da su podaci razlog postojanja programa. Svaki postupak obrađuje neke podatke (čita ih, prikazuje, menja,...). Podaci i postupci nad njima predstavljaju jednu nerazdvojnu celinu, jer nam podaci bez postupaka kojim ih obrađujemo, kao i postupci za obradu bez podataka, ne znače puno. Kod objektno orijentisanih jezika podaci i postupci su objedinjeni u jednu celinu i čine aktivan objekat, za razliku od podataka u proceduralnim jezicima koji su krajnje pasivni. Svaki takav objekat izgleda kao mali računar: on ima unutrašnje stanje i ima operacije koje možemo zahtevati da izvrši. Jasno možemo uočiti analogiju između takvih objekata i objekata u realnom svetu.

 Posmatrajmo niz imena učenika jednog odeljenja. Kod proceduralnog programiranja niz imena se posmatra kao skup pasivnih podataka. Potom pišemo razne programe za rad sa ovim nizom u cilju njegovog abecednog uređivanja, modifikovanja (npr. dodavanje novog učenika, brisanje postojećeg učenika, ...). Sa jedne strane imamo podatke, a sa druge od njih odvojene postupke(procedure) kojima obrađujemo te podatke. Kod objektno orijentisanih jezika, prethodni primer posmatra se objedinjeno: postoji objekat niz koji sa jedne strane ima skup imena, a sa druge strane skup akcija koje se mogu primenjivati nad tim imenima i jedno od drugog je neodvojivo. Program koji koristi objekat niz ne mora imati ni jedan od postupaka za obradu niza jer su oni ugrađeni u sam objekat.

Mnoštvo različitih objekata možemo uočiti u okruženju grafičkih operativnih sistema (GUI- Graphic User Interface). Ikone su objekti. Svakoj je pridružen skup osobina i akcija koje opisuju kako reaguju na spoljašnje događaje - šta se dešava kada jednom kliknemo, kada dva puta kliknemo, kad pritisnemo desni taster, kako se ponašaju kada držimo pritisnut taster na mišu i slično.

U objektno orijentisanom programiranju cilj programera je da sakupi podatke i akcije u neku vrstu „kutije sa alatom“ koju će koristiti za brzi razvoj aplikacija. Primeri objektno orijentisanih jezika su SmallTalk, Java, C#. C++ je hibridni jezik; jeste iz klase objektno orijentisanih jezika ali nije čisto objektno orijentisan.

Iz ovog kratkog prikaza razvoja programskih jezika možemo uočiti da objektno orijentisanim pristupom na prirodan način opisujemo realan svet. Ipak prirodnost opisa nije glavni razlog ozbiljnijeg razvoja objektno orijentisanih jezika, već je to „softverska kriza“ nastala osamdesetih godina. U to vreme, razvojem hardvera, došlo je do znatnog povećanja primene računara u raznim oblastima, pa su samim tim čitavi timovi programera morali da rade na razvoju jednog projekta. To je realizovano tako što svaki deo tima razvija relativno nezavisne module, a zatim se ti moduli uklapaju u funkcionalnu celinu, korisničku aplikaciju. Svaki od modula može se koristiti u različitim aplikacijama tako da na taj način dolazi do standardizacije aplikativnog softvera.

S druge strane, pored razvijanja objektno orijentisane metodologije u programiranju, razvojem Interneta pojavila se i potreba da napisani projekti identično rade na različitim platformama (operativni sistemi,...). Razvojem programskog jezika Java, kao primera objektno orijentisanog jezika, na neki način se prevazilaze različite platforme na kojima bi se softver izvršavao. Java tehnologija izvorno je bila zamišljena za prenos multimedije mrežom raznovrsnih uređaja.

Ponovimo ukratko:

Program u proceduralnom jeziku je niz naredbi koji određuje KAKO se određena akcija obavlja (npr. „otvori datoteku, ako nije EOF pročitaj podatak, ..., zatvori datoteku“)

Program u neproceduralnom jeziku je niz naredbi koji određuje ŠTA treba učiniti npr. „izdvoji podatke … koji zadovoljavaju dati uslov …“

Program u objektno orijentisanom jeziku možemo shvatiti kao skup objekata koji između sebe i sa spoljnim svetom komuniciraju putem „poruka“.

Algoritmi

Zbog toga je došlo do razvoja programskih jezika koji omogućavaju zapis niza instrukcija računaru koje se ne mogu višeznačno protumačiti. Jednoznačnost svake konstrukcije programskog jezika je njegova najbitnija karakteristika. Programski jezik je skup pravila kojim se predstavljaju instrukcije i opisuju podaci računaru.

Rešavanje problema primenom računara

Da bismo rešili neki problem korišćenjem računara neophodno je kreirati program u nekom programskom jeziku. Taj proces je kompleksan i u njemu možemo izdvojiti sledeće faze:

Postavka problema

Problem koji se rešava treba precizno i potpuno opisati na prirodnom jeziku. To radi korisnik kome je namenjen program u saradnji sa programerima. Jasno se određuje cilj, analiziraju se postojeće informacije i vrši se izbor neophodnih početnih informacija, opisuju se svi podaci.

Analiza problema

U ovoj fazi definišu se ulazni i izlazni podaci, ograničenja njihovih vrednosti i precizno definišu veze između podataka. Rezultat ove faze je formalan opis problema, izrada matematičkog modela koji se može realizovati na računaru.

Razrada algoritma

U ovoj fazi se definiše način na koji se od početnih – ulaznih podataka dolazi do traženih - izlaznih. Većina problema može da se reši na više načina, zato treba sve načine proanalizirati i izabrati optimalan. Složenost rešenja problema možemo posmatrati sa aspekta broja potrebnih operacija (vremena potrebnog za izvršavanje algoritma), i sa aspekta potrebnih memorijskih resursa da se algoritam izvrši (prostorna složenost algoritma). Dokaz o korektnosti izabranog postupka za rešavanje problema treba i precizno izvesti. O ovim aspektima treba posebno voditi računa kada rešavamo složene probleme, dok kod jednostavnih problema ih zanemarujemo.

 Projektovanje opšte strukture programa

U ovoj fazi vrši se izbor programskog jezika. Razmatra se ceo problem i deli se u logičke celine, svaka celina se dalje deli u zavisnosti od njene složenosti. Precizno se definiše način čuvanja informacija tj. definiše se struktura podataka.

Kodiranje

U ovoj fazi opisuju se, u nekom programskom jeziku, podaci i postupak za rešavanje problema, koji su definisani u prethodnim fazama. Program zapisan u nekom programskom jeziku često se zove kod, zato se ova faza i naziva kodiranje. Ako smo u prethodnim fazama precizno razradili algoritam, strukturu programa i strukturu podataka, ova faza je relativno jednostavna za izvođenje.

Faza prevođenja, izvršavanja i testiranja programa

Podsetimo se da na računaru može da se izvršava samo program koji je napisan pomoću binarnih cifara (mašinskom jeziku). Zato je potrebno program napisan na programskom jeziku višeg nivo prevesti na mašinski jezik, prevođenje se vrši pomoću posebnih programa kompajlera odnosno interpretera. Da bi se program izvršavao potrebno je formirati izvršnu verziju (EXE verzija), to se postiže pomoću specifičnih programskih alata koji se razlikuju od jezika do jezika (povezivač, bilder,...). Često su faza prevođenja i faza pravljenja izvršnog fajla objedinjene. Testiranjem programa treba proveriti da li program rešava postavljeni zadatak. To je vrlo važna faza u kojoj treba otkriti eventualno skrivene greške. Za otkrivanje grešaka koristimo testove sa kojima treba da obuhvatimo sve moguće situacije, svakako i one kad postavljeni problem nema rešenja. Po potrebi se vraćamo na prethodne faze.

Izrada dokumentacije

U dokumentaciji programa opisuje se šta program radi, daju se uputstva kako se koristi, detaljnije se opisuje algoritam kada su u pitanju složeniji problemi i slično. Dokumentaciju nikako ne treba započinjati kada je program već napisan već je pisati tokom izrade programa, ali je svakako treba dopuniti na kraju kada je program završen.

Održavanje programa

Tokom korišćenja programa mogu se uočiti neke greške, koje svakako treba otkloniti. Ponekad treba izvršiti neke promene u programu zbog novih okolnosti. Ako je program čitko napisan i ako ima dobru dokumentaciju onda je ova faza jednostavna, kako za same autore programa, tako i za druge programere, u suprotnom mnogim programerima je lakše da napišu novi program nego da prepravljaju već postojeći. U ovoj fazi dolazi i do proširivanja program u pravcu dodavanja novih funkcionalnosti

Kada se razvija veliki projekat sve prethodno navedene faze su jednako važne. Osposobljavanje programera za uspešno rešavanje velikih projekata je dugotrajan proces. U edukaciji polazimo od jednostavnih primera ka složenijim. U ovom procesu problemi koje rešavamo su jednostavni pa su sve faze osim faze razrade algoritma jednostavne.

Pojam i karakteristike algoritma

Termin algoritam potiče od imena Arapskog matematičara Al-Horezmija (oko 825 godine nove ere). On je izvršavanje aritmetičkih operacija prikazivao u obliku uputstava koja se sastoje od osnovnih koraka. Takav način opisivanja rešenja od tada se u matematici naziva algoritam. Daljim razvojem nauke algoritmi su postali jedan od osnovnih pojmova u matematici i računarstvu.

Algoritam je konačan niz jednostavnih koraka kojim se opisuje postupak rešavanja određenog realnog problema, ali takav da se posle konačno mnogo vremena postupak završava.

Mnoge realne situacije se odvijaju po tačno određenim koracima tako da možemo reći da su algoritmi sastavni deo svakodnevnog života. Postoji relativno precizan postupak po kome pokrećemo automobil, telefoniramo, prelazimo ulicu, kuvamo kafu… Čak možemo reći da se i sam život odvija po nekim pravilima. Dobar primer algoritma je svaki kulinarski recept, u njemu jasno razlikujemo šta je ulaz (potrebni sastojci), šta je obrada (precizno opisan postupak za kuvanje) i šta je izlaz (gotovo jelo).

Svaki algoritam mora da ima sledeća svojstva:

definisanost

*svaki korak u algoritmu mora biti jednoznačno definisan*

determinisanost

*vrednost koja se dobija posle izvršavanja svakog koraka jednoznačno je određena vrednostima iz prethodnog koraka*

konačnost

*rad algoritma mora se završiti u konačnom broju koraka*

rezultativnost

*algoritam mora dati rezultat za sve situacije za koje je kreiran*

masovnost

*algoritam treba da obezbedi rešavanje cele grupe problema koji se razlikuju samo po ulaznim veličinama.*