OOP

PREDAVANJE 1

Prof. dr Borivoje Milošević

2018. godina

Sadržaj, ciljevi i preduslovi

- Sadržaj:
  - Uvod
  - Osnovni koncepti OOP (Object Oriented Programming) i objektne paradigme
  - Jezik C++
  - Laboratorijske vežbe
- Ciljevi:
  - Upoznati se sa osnovnim konceptima objektne paradigme i njihovim pogodnostima
  - Osposobiti se za projektovanje i pisanje programa korišćenjem objektnih koncepata
  - Osposobiti se za programiranje na jeziku C++

- Preduslovi:
  - Dobro poznavanje osnovnih principa i koncepata proceduralnog programiranja
  - Poznavanje rada sa računarom i osnovnim alatkama za programiranje (editor, prevodilac, debager)
  - Dobro savladano gradivo predmeta Osnovi programiranja, sa praktikumom iz programiranja
  - Veština u programiranju na jeziku C
  - **Samostalan, praktičan i kontinualan rad!**

Organizacija nastave i praktičan rad

- Predavanja: 2 časa nedeljno
- Kolokvijumi: Ukupno 2 u semestru
- Računske vežbe:
  - 1 čas nedeljno
  - zadaci za razumevanje koncepata i uvežbavanje jednostavnih programerskih principa
  - diskusija, demonstracije i konsultacije
- Laboratorijske vežbe:
  - 1 čas nedeljno
  - rešavanje zadataka u vezi sa nastavnim temama upotrebom računara na platformi Visual C++
  - diskusija, demonstracije i konsultacije
- Praktičan samostalan rad:
  - domaći zadaci u toku semestra
  - rade se i brane samostalno, usmeno i na računaru
  - ulaze u konačnu ocenu

LITERATURA:

BORIVOJE MILOŠEVIĆ “ Programski jezici 2 “, Niš, SIKC, 2016.

DRAGAN MILIĆEV, “Objektno orijentisano programiranje na jeziku C++”, Mikro knjiga Beograd, 2001.

JESSE LIBERTY, “ Naučite C++”, Kompjuter biblioteka, 1998., Čačak

LASLO KRAUS,” Programski jezik C++”, Akademska misao, Beograd, 2001.

MILENA STANKOVIĆ, “ Programski jezici”, Elektronski fakultet, Niš, 2000.

MAGDALINA TODOROVA, “ Programiranje na C++”, Siela Soft and Publishing, Sofia, Bulgaria, 2002.

C++

C++ JE NADSKUP JEZIKA C

Bjarne Stroustrup
etalon za C++

Professor and holder of the College of Engineering Chair in Computer Science
Texas A&M University

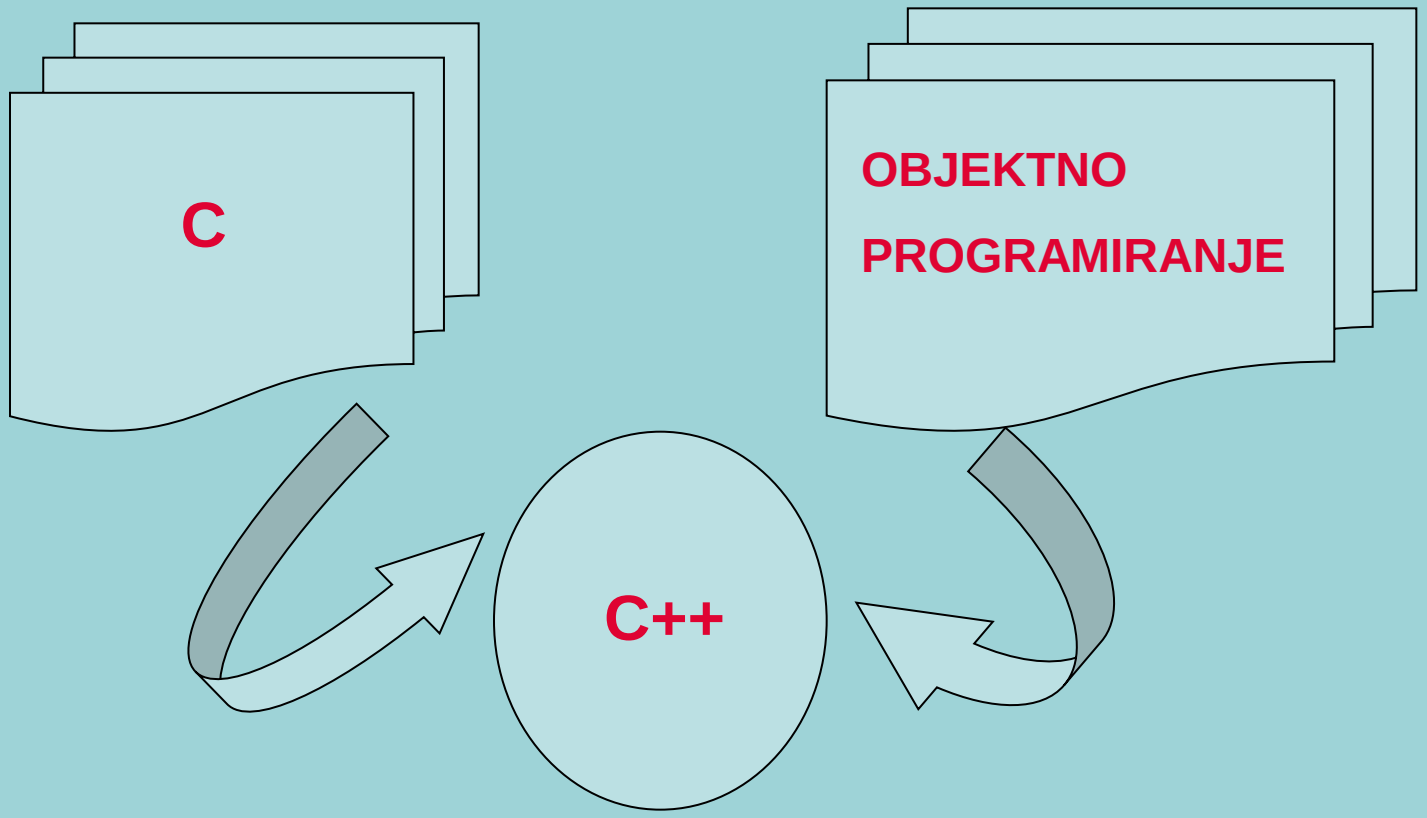
The C++ Programming Language and The Design and Evolution of C++.

- **Bjarne Stroustrup** (rođen 11. juna 1950. u Arhusu, Danska) je magistar je nauka (matematika i informatičke nauke), 1975, Univerzitet u Arhusu u Danskoj.

- Doktorirao kompjuterske nauke 1979. god. na Kembridž univerzitetu u Engleskoj. Dizajnirao i implementirao C++ programski jezik. Autor je više knjiga o programskom jeziku C++.

POVEZUJE

ANSI STANDARD 1997



“THE AUTOADDED C++ REFERENCE MANUAL”

from MARGARET ELLIS and BJARNE STROUSTRUP 1991 GOD.

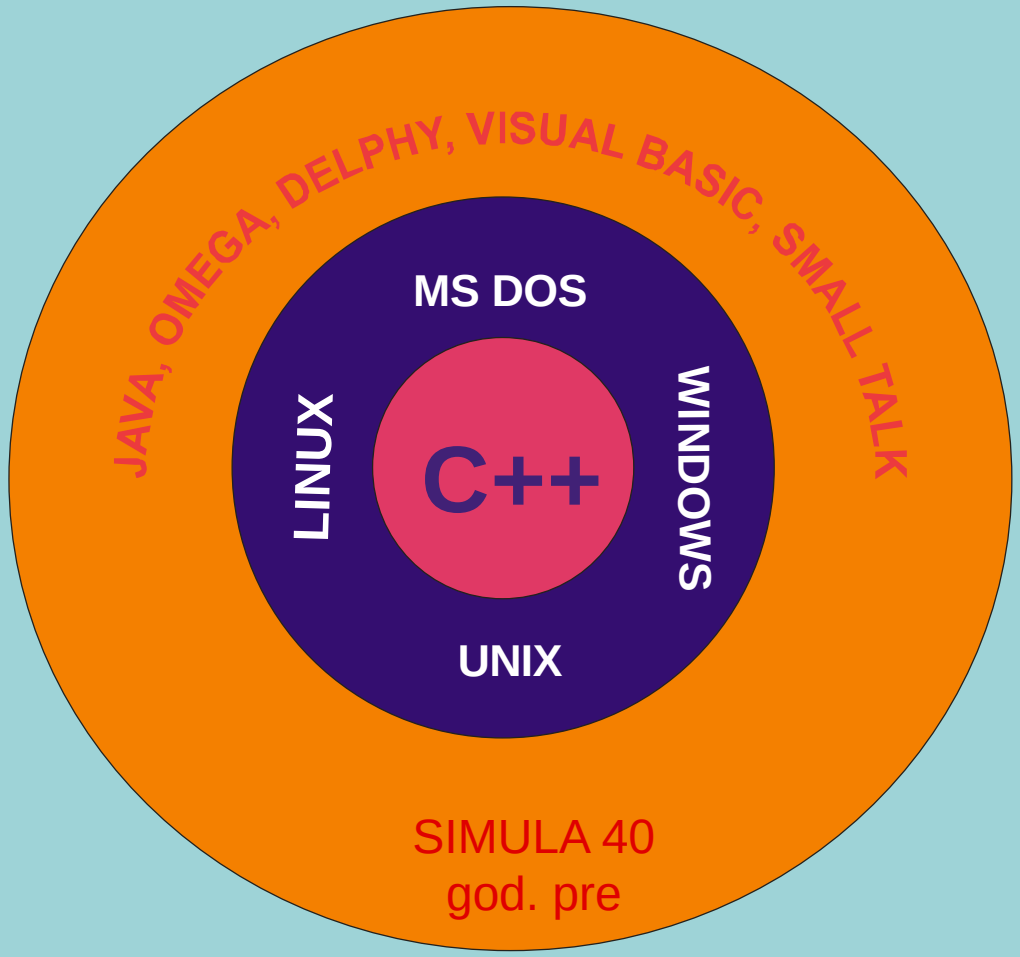
VIŠESTRUKO NASLEĐIVANJE

APSTRAKTNE KLASE

MEHANIZMI GENERIČKIH KLASA

IZUZECI I GREŠKE

OPERATIVNI SISTEMI I OBJEKTNO ORIJENTISANI JEZICI

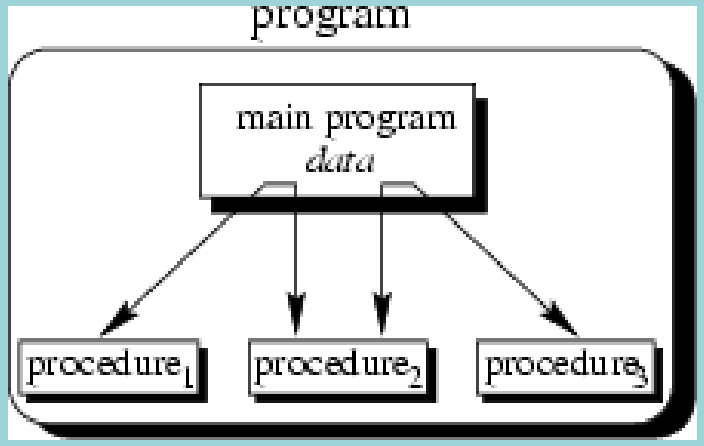


Pregled tehnika programiranja

- **Nestrukturno - proceduralno programiranje**
  - **Glavni program direktno operiše sa globalnim podacima.**
  - **Dugi i nepregledni programi**
  - **Copy paste - Kod se višestruko koristi kopiranjem delova**

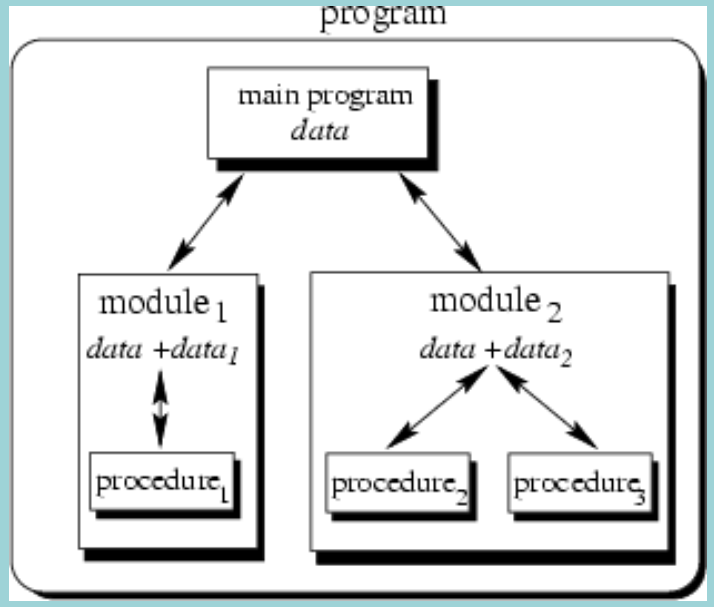
Pregled tehnika programiranja

- **Proceduralno programiranje**
  - **Program se može posmatrati kao sekvenca poziva potprograma (procedura).**
  - **Strukture podataka se modeliraju odvojeno od koda procedura koje ih obrađuju.**
  - **Višestruko korišćenje koda postiže se preko biblioteka procedura i funkcija.**



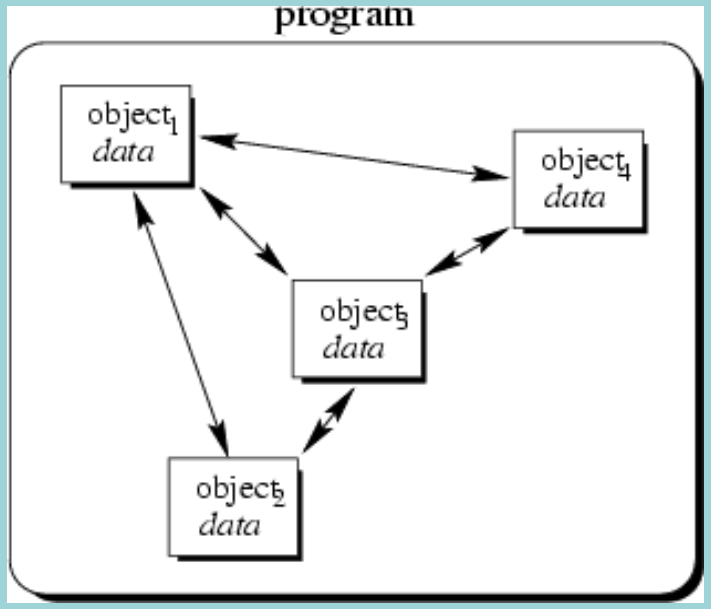
Pregled tehnika programiranja

- **Modularno programiranje**
  - **Procedure sa zajedničkom funkcionalnošću su integrisane u jedan modul**
  - **Svaki modul može da ima svoje sopstvene podatke.**
  - **Višestruko korišćenje struktura podataka i procedura**



Pregled tehnika programiranja

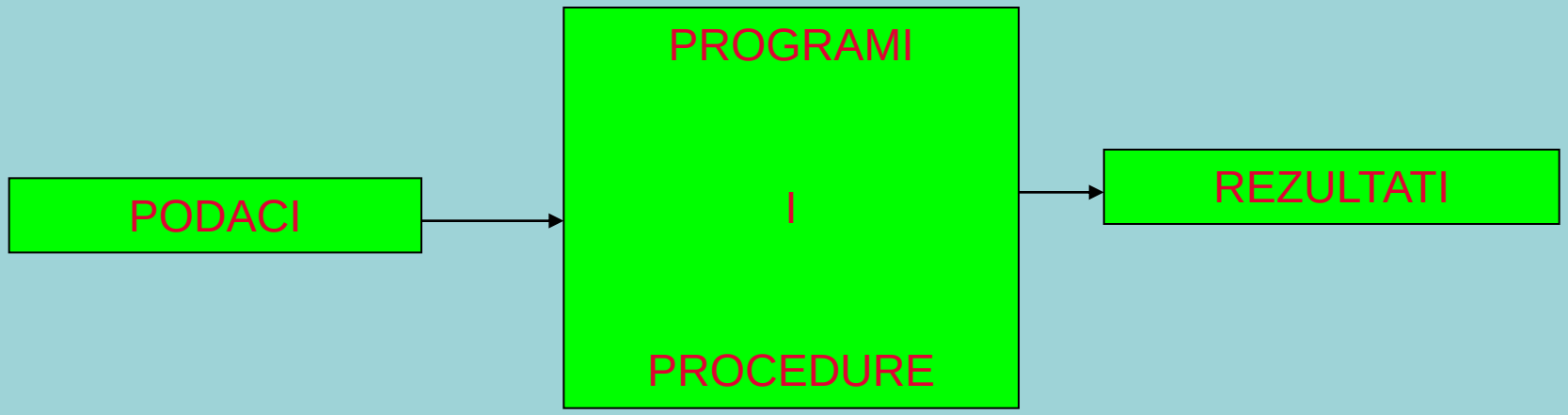
- **Object Oriented Programming**
  - **Strukture podataka i procedure integrisane u klase**
  - **Program može da se posmatra kao mreža objekata koji su u interakciji pri čemu svaki objekat ima svoje stanje.**
  - **Apstracija, inkapsulacija, nasleđivanje i polimorfizam**
  - **Ponovno korišćenje objekata**



Pregled tehnika programiranja

- **Komponentno programiranje (Component Development)**
  - **Aplikacija se gradi od prekompajliranih softverskih blokova, slično konceptu integrisanih kola u elektronici**
  - **Akcenat je na modeliranju interfejsa**
  - **Komponente se mogu višestruko koristiti (Binary reuse of functionality)**
  - **Nezavisnost od programskog jezika i platforme na kojoj se aplikacija koristi.**

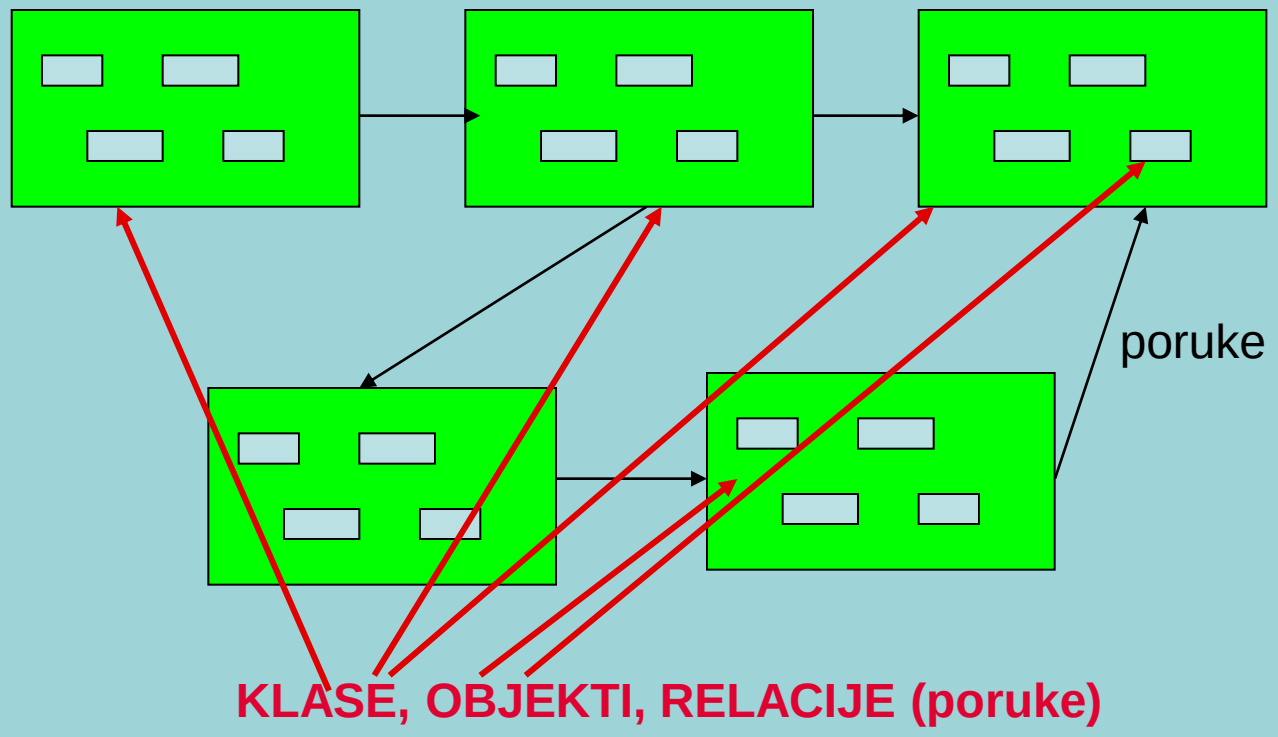
BLOK DIJAGRAM PROCEDURALNOG PROGRAMIRANJA



Procedural	Object-oriented
procedure	method
record	object
module	class
procedure call	message

BLOK DIJAGRAM OBJEKTNOG PROGRAMIRANJA

“ Objektno orijentisano programiranje je paradigma programiranja, koja koristi objekte kao osnovu za projektovanje računarskih programa i različitih aplikacija softvera. Zasniva se na različitim tehnikama, kao što su nasleđivanje, modularnost, polimorfizam i enkapsulacija. "



OBJEKTNO PROGRAMIRANJE

- Rešavanje problema paradigmom objektno-orijentisanog programiranja je vrlo slično ljudskom načinu razmišljanja i rešavanju problema iz realnog sveta.
- Sastoji se od identifikovanja objekata i postavljanju objekata koji će se koristiti u odgovarajućoj sekvenci za rešenje određenog problema. Radi se o dizajnu objekata čija će ponašanja kao jedinica i u njihovoj međusobnoj interakciji, rešiti određeni problem.
- Interakcija između objekata se sastoji u razmeni poruka, gde određena poruka usmerena prema određenom objektu, pokreće enkapsulirane operacije u tom objektu, čime se rešava deo obično šireg i složenijeg problema. Uopšteno gledano, objektno-orijentisano rešavanje problema se sastoji iz četiri koraka:

- identifikovanje problema
- identifikovanje objekata koji su potrebni za njegovo rešenje
- identifikovanje poruka koje će objekti međusobno slati i primati
- kreiranje sekvence poruka objektima, koje će rešavati problem ili probleme.

OBJEKTNO PROGRAMIRANJE

- Prilikom rešavanja nekog problema, objektni programer najviše će vremena posvetiti analizi i modelovanju problema, sa ciljem da identifikuje potrebne elemente, tzv. entitete i njihove medjusobne veze u okviru domena problema.
- Da bi se dobilo uredjeno znanje o nekom domenu problema, entiteti se grupišu u kolekcije koji se nazivaju klase entiteta.
- Terminom entitet se označava da nešto postoji, a to nešto ima neka svojstva i stoji u nekom odnosu sa drugim entitetima. Termin entitet se uveliko ustalio u prirodnim i inženjerskim naukama.
- Klasa entiteta je kolekcija entiteta, koji su prema nekom kriterijumu, medjusobno slični.

OBJEKTNO PROGRAMIRANJE

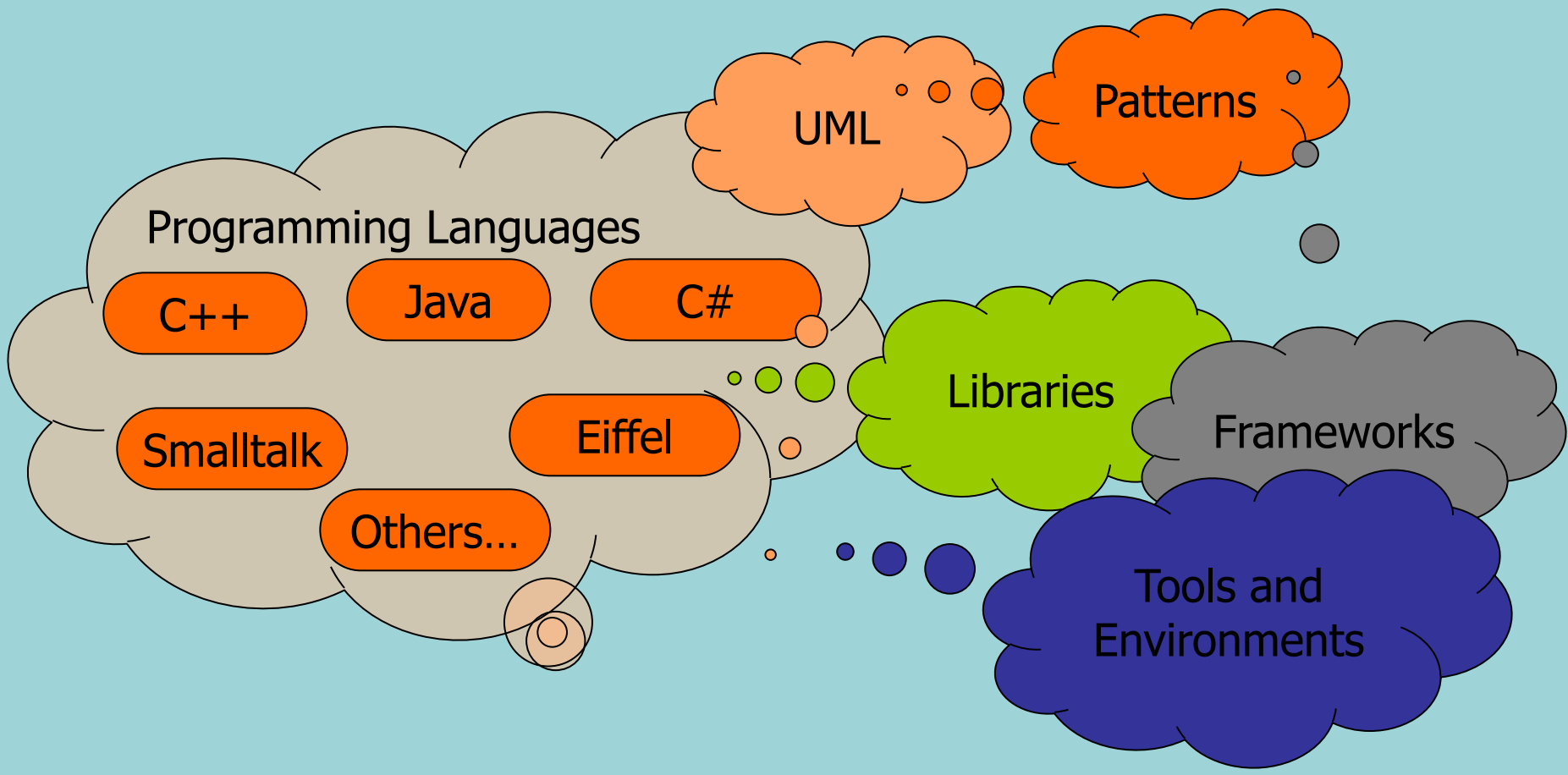
- U paradigmi objektno orijentisanog programiranja, objekti su strukture podataka koje predstavljaju određeno i jasno definisano znanje o spoljašnjem svetu ili stvarnosti. Tipična organizacija je u hijerarhijske klase, gde svaka klasa objekta poseduje informacije o osobinama objekta koje se čuvaju u instancama promenljivih i koje su povezane (konceptom asocijacije) sa svakom instancom u određenoj klasi.

- Svaki objekat prepoznaje drugi objekat preko njegovog interfejsa. Podaci i logika svakog objekta su skriveni od drugih objekata. Time se omogućava razdvajanje implementacije od ponašanja objekta u interakciji sa drugim objektima.

- Osnovne osobine objekata su identitet, stanje i ponašanje.

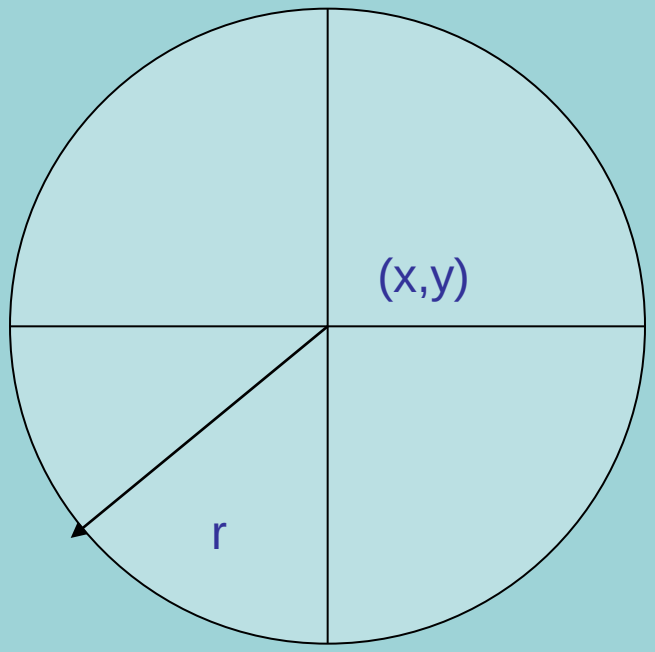
  - Identitet predstavlja naziv objekta kojim se određeni objekat razlikuje od ostalih.
  - Stanje objekta je deo prošlosti i sadašnjosti koje određuju ponašanje objekta u budućnosti.
  - Ponašanje objekta je određeno operacijama koje se nad objektom mogu izvršiti, a aktiviranje operacije se vrši porukom.

Šta čini OO tehnologiju?



PRIMER:

- Napisati program na nekom od Objektno orijentisanih jezika za crtanje kruga sa krstom na sredini.



REŠENJE:

- Da bi nacrtali traženi crtež, potrebna su tri objekta:

- KRUG: krug pripada klasi CIRCLE
- VERT: vertikalna linija pripada klasi LINE
- HORI: horizontalna linija pripada klasi LINE

Za svaku klasu postoji elementaran konstruktor koji se može koristiti za kreiranje objekata sa nekom inicijalnom strukturom. Za inicijalizaciju novokreiranih objekata mogu se definisati konstruktori sa istim imenom a različitim brojem i tipom argumenata. Tada sledi, koristeći C++ notaciju:

- KRUG = new CIRCLE (x,y,r)
- VERT = new LINE (x,x-r,x,y+r)
- HORI = new LINE (x-r,y,x+r,y)

- KRUG → display ( )
- VERT → display ( )
- HORI → display ( )

ZAŠTO OOP ?

- SOFTVERSKA KRIZA

  - ✓ **Drastično povećanje zahteva korisnika jer su programeri pokazali šta sve računari mogu da rade.**

  - ✓ **Povećanje korisničkih zahteva uslovljava povećanu produktivnost programera (ili njihovog broja), ali dostupno proceduralno programiranje tada je dopuštalo projektovanje softvera u modulima sa relativno jakom interakcijom – svaka promena u programu mora biti izvršena u svim modulima.**

  - ✓ **Povećanje produktivnosti moguće je ponovnim korišćenjem delova softvera koji je već ranije urađen i korišćen, ali bez mnogo dorade za nove stvari i rešenja (slaba interakcija) što predstavlja veliki problem korišćenjem standardnog načina programiranja.**

  - ✓ **Drastično povećanje troškova održavanja softvera utiču na to da softver mora biti čitljiviji, lakši za nadgradnju i modifikovanje i samo održavanje.**

Objektno orijentisano programiranje

- **OO karakteristike**
  - **Sve je objekat.**
  - **Objekat je definisan podacima i skupom metoda koji opisuju njegovo ponašanje.**
  - **Objekti komuniciraju međusobno slanjem poruka.**
  - **Svaki objekat pripada svojoj klasi (ima svoj tip).**
  - **Klasa opisuje strukturu objekta i njegovo “ponašanje”.**
  - **Svi objekti iste klase mogu da odgovore na iste poruke.**

Objektno orijentisano programiranje

- **Osnovni koncepti**
  - **Klasa**
    - Definiše strukturu objekata (podatke) i njihovu funkcionalnost (ponašanje). Klasa može da se posmatra kao šablon objekata, (template) kojim se opisuje model po kome ae se kreirati novi objekat. Klase se ponekad nazivaju fabrikama objekata (Cox 1986)
  - **Objekat**
    - Primerci (istance) klasa, pojedninačne pojave svake klase.
    - Svi objekti jedne klase imaju strukturu definisanu klasom i nad njima se mogu izvršavati samo operacije definisane klasom kojoj pripadaju.
  - **Poruka**
    - Objekat odgovara na poruku time što se izvršava odgovarajući metod.
    - Kažemo da na objekte delujemo porukama, pri čemu svaka poruka pretstavlja poziv metoda (funkcije ili procedure) definisanog u klasi kojoj objekat pripada.

Objektno-orijentisano programiranje

- Prema Alan Kay-u, **čisto pristup objekt orijentisanom programiranju poseduje pet esencijalnih karakteristika:**
- **Sve je objekat.** Posmatrajmo objekat kao sofisticiranu promenljivu koja omogučava: uskladištenje vrednosti i podršku operacijama nad tom vrednošću.
- **Program se transformiše u skup objekata koji sarađuju razmenom poruka**. Poruku možemo posmatrati kao mehanizam poziva operacije nad objektima.
- **Svaki složeni objekat je izveden kao kolekcija objekata**.
- **Svaki objekat pripada određenom tipu tj. predstavlja instancu KLASE (tip = klasa)**.
- **Svi objekti iste klase mogu primati iste poruke – polimorfizam**.

Objektno orijentirano programiranje OOP

- **OOP je način ugradnje programske podrške kod kojeg su programi organizovani kao kooperativni skupovi objekata, od kojih svaki predstavlja pojavu neke klase, a sve klase su članovi neke hijerarhije klasa sjedinjene preko nasleđivanja**
  - **Programiranje bez nasleđivanja nije objektno orijentisano, već je reč o programiranju sa apstraktnim tipovima podataka**
- **Osnovni gradivni element jezika je objekt a ne algoritam**

Jezik je objektno orijentisan ako i samo ako:

- **podržava objekte koji predstavljaju apstrakciju stvarnog sveta sa skrivenim lokalnim stanjem i sa interfejsom koji defini[e operacije nad tim objektima**
- **objekti pripadaju nekoj klasi (class)**
- **klase mogu nasleđivati svojstva od nadklasa**

- **Za jezik bez nasleđivanja kaže se da je zasnovan na objektima (object-based)**

OOP uvodi drugačiji način razmišljanja u programiranje!

- **U OOP-u, *mnogo* više vremena troši se na *projektovanje,* a mnogo manje na samu implementaciju (kodovanje).**
-
- **U OOP-u, razmišlja se najpre o *problemu,* a tek naknadno o programskom rešenju.**
- **U OOP-u, razmišlja se o delovima sistema (objektima) koji nešto rade, a ne o tome kako se nešto radi (algoritmima). Drugim rečima, OOP prvenstveno koristi *objektnu dekompoziciju* umesto isključivo *algoritamske dekompozicije.***
-
- **U OOP-u, pažnja se prebacuje sa realizacije na medusobne veze između delova. Teži se što većoj redukciji i strogoj kontroli tih veza. Cilj OOP-a je da smanji interakciju između softverskih delova.**

OOP uvodi drugačiji način razmišljanja

- **Posmatrajmo na primer izraz “log 1000”. Da li je ovde u središtu pažnje funkcija “log” ili njen argument “1000”?**
- **Ukoliko razmišljamo da se ovde radi o funkciji “log” kojoj se *šalje* argument “1000” i koja *daje* kao rezultat broj “3”, razmišljamo na proceduralni način.**
- **Sa druge strane, ukoliko smatramo da se u ovom slučaju na broj “1000” *primenjuje* operacija (funkcija) “log” koja ga *transformiše* u broj “3”, tada razmišljamo na objektno orijentisani način.**
- **Dakle, proceduralno razmišljanje forsira *funkciju*, kojoj se *šalje* podatak koji se obrađuje, dok objektno orijentisano razmišljanje forsira *podatak* nad kojim se funkcija *primenjuje*.**

OOP uvodi drugačiji način razmišljanja

Navedimo još jedan primer.

- **Posmatrajmo izraz “5 + 3”. Proceduralna filozofija ovdje stavlja središte pažnje na *operaciju* sabiranja (“+”) kojoj se kao argumenti *šalju* podaci “5” i “3”, i koja *daje* kao rezultat “8”.**

- **Međutim, sa aspekta objektno orijentisane filozofije, u ovom slučaju se nad podatkom “5“ *primjenjuje* akcija “+ 3” (koju možemo tumačiti kao “povećaj se za 3”) koja ga *transformiše* u novi podatak “8”.**

- **Međutim ne može se reći da je bilo koja od ove dve filozofije ispravna a da je druga neispravna. Obe su ispravne, samo imaju različita gledišta.**
- **Međutim, praksa je pokazala da se za potrebe razvoja složenijih programa objektno orijentisana filozofija pokazala znatno efikasnijom, produktivnijom, i otpornijom na greške u razvoju programa.**

Prednosti OOP

- **Sposobnost jednostavnog korišćenja delova koda u različitim programima, štedeći vreme u analizi, dizajnu, razvoju, testiranju, i debagovanju,**

- **Sposobnost kupovine delova postojećeg, testiranog koda da bi se u razvoju omogućio komponentno-zasnovani pristup, smanjeni troškovi razvoja,**

- **Sposobnost jednostavne podele velikih programerskih projekata između razvojnih timova, poboljšane karakteristike debagovanja i testiranja, viši kvalitet softvera,**

- **Sposobnost da se programski zadaci, koji su preveliki da bi se rešili u jednom kratkom računu, podele i reše,**

- **Sposobnost kreiranja jednostavnih i konzistentnih sredstava za međudelovanje različitih tipova objekata, pri čemu se krije kompleksnost razlika koje postoje «iza zavjese».**

- **Svi programski jezici obezbeđuju podršku određenim apstraktnim kategorijama**.

- **Asemblerski jezik** poseduje nizak nivo apstrakcije u odnosu na fizičku mašinu (hardware).

- Većina **imperativnih programskih jezika** (FORTRAN, BASIC, C) su samo apstrakcije asemblerskog jezika pa, iako predstavljaju značajan napredak u odnosu na asemblerski jezik, u osnovi njihove apstrakcije leže kategorije koje odgovaraju arhitekturi računarskog sistema a ne problema koji se rešava.
  U ovom slučaju, zadatak programera je da uspostavi korespondenciju između domena rešenja (apstraktnog modela mašine) i domena problema (apstraktnog modela realnog sveta).
  Napor koji je neophodno uložiti u ovo preslikavanje, a koji u jednom delu leži u samoj strukturi korišćenog programskog jezika,
  često nema adekvatnu valorizaciju budući da rezultuje programima niskog stepena semantike.

Alternativa prethodno opisanom pristupu je modeliranje problema koji se rešava.

Prvi programski jezici koji su razvijeni na problemskoj orijentaciji uvode specifične apstrakcije realnosti na kojima zasnivaju postupak preslikavanja domena problema u domen rešenja.

Tipični primeri su sledeći pristupi:

- Sve probleme je moguće **apstrahovati listom i operacijama nad njom (LISP – LISt Processing).**
- Svi problemi su **algoritamske prirode (APL-Algoritmic Programming Language).**
- Svi problemi se daju **iskazati kao lanci odlučivanja (PROLOG – PROgramming LOGic).**
- Svi problemi se mogu iskazati kao **skup apstraktnih ograničenja i manipulacija sa njima (Constraint Based Programming).**

Svi navedeni pristupi predstavljali su dobra rešenja za određenu usku klasu problema za koju su objektivno i dizajnirani, ali svaki pokušaj generalizacije iskoraka van inicijalnog domena gotovo bez izuzetka rezultuje neuspehom.

- **Objektno-orijentisani pristup** posebnim čini obezbeđenje alata i mehanizama za iskazivanje problema pojmovima iz domena problema.
  Apstrakcije koje ono uvodi su dovoljno opšte tako da programer nije ograničen jednim domenom primene. Sada postaje **bitno šta uraditi a ne kako uraditi.**

C++

- **C++ nije čisti OOP jezik jer se može koristiti “kao malo bolji” C.**
- **C++ uvodi sasvim drugačiji način programiranja**
- **C++ više vremena troši na projektovanje rešenja a manje na implementaciju (kodiranje)**
- **C++ razmišlja najpre o problemu a tek posle o programskom rešenju**
- **C++ razmišlja o delovima sistema (objektima) koji nešto rade a ne o tome kako se nešto radi. Koristi se *objektna dekompozicija* umesto *algoritamske dekompozicije*.**
- **C++ pažnju prebacuje sa relacija na međusobne veze između delova (veća redukcija i stroga kontrola tih veza) a smanjuje se interakcija između softverskih delova programa.**

Objektno-orijentisano programiranje

Predstavlja implementacionu metodu kod koje su programi organizovani kao kooperativni skup objekata pri čemu svaki objekat predstavlja instancu neke klase iz hijerarhije klasa nastale na bazi relacija nasleđivanja (inheritance).

Objektno-orijentisana analiza

Predstavlja analitičarsku metodu koja zahteve (requirements) posmatra iz perspektive **KLASA i OBJEKATA** koji pripada domenu problema.

Objektno-orijentisani dizajn

Predstavlja projektantsku metodu koja obuhvata proces objektno orijentisane dekompozicije i notaciju za iskazivanje logičkih i fizičkih kao i statičkih i dinamičkih modela posmatranog sistema.

Elementi objektnog modeliranja

Razlikujemo četiri osnovna:
APSTRAKCIJA (Abstraction)
ENKAPSULACIJA (Encapsulation)
NASLEĐIVANJE (Inheritance)
POLIMORFIZAM (Polymorphism)

i tri sporedna:
TIPIZACIJA (Typing)
KONKURENCIJA (Concurency)
PERZISTENCIJA (Persistency)
elementa objektnog modeliranja.

Elementi objektnog modeliranja

APSTRAKTNI TIPOVI PODATAKA
ABSTRACT DATA TYPES

Pored (int, float, char) proizvoljno se definišu tipovi (complex, point, disk, jabuka) nad kojima korisnik vrši operacije -multiple instances-

NASLEĐIVANJE
INHERITANCE

Od klase npr. Printer sa svojim operacijama printline, lineFeed, formFeed, dovoljno je formirati tip PrinterWithFont koji nesleđuje sve osobine starog

ENKAPSULACIJA
ENCAPTULATION

Definiše šta se sa tipom može raditi a način kako se to radi "sakriva" se od korisnika

POLIMORFIZAM
POLYMORPHISM

Raniji delovi programa koji koriste Printer ne moraju se prepravljati jer će jednako dobro raditi i sa tipom PrinterWithFont, čak se ne moraju prevoditi

Apstraktni tipovi podataka - klase, objekti i poruke

- **Apstrakcija podataka predstavlja prvi korak ka objektno-orijentisanom programiranju.**
- **Ključni pojmovi u ovim jezicima su:**
  - **pojam objekta i**
  - **pojam klase.**

  Klasa je slična apstraktnom tipu podataka po tome što definiše strukturu objekata i eksterni interfejs, funkcije i procedure (u terminologiji objektnih jezika metode) kojima se može delovati na objekte. *Može se reći da klasa opisuje strukturu i ponašanje objekata*. Objekti su pojedninačni primerci, instance klase. Svi objekti jedne klase imaju strukturu definisanu klasom i nad njima se mogu izvršavati samo operacije definisane klasom kojoj pripadaju.

Apstrakcija

- **Bitan element objektno orijentisanih jezika. Činjenica je da se čovek bori sa složenim situacijama apstrahovanjem.**
  - **Na primer, niko ne zamišlja automobil kao skup desetina hiljada pojedinačnih delova, već ga prihvata kao dobro definisan objekat koji se ponaša na jedinstven način.**
- **Ovakvo apstrahovanje omogućuje nam da sednemo u automobil i obavimo kupovinu ne opterećujući se složenošću mehanizma automobila. Na taj način možemo da zanemarimo detalje rada motora, prenosnog mehanizma i kočnica i da objekat koristimo kao celinu.**

Apstrakcija

- **Siguran način za snalaženje u apstrakcijama predstavlja korišćenje hijerarhijske klasifikacije. To nam dozvoljava da semantiku složenih sistema uredimo po slojevima, razdvajajući ih na celine kojima se lakše upravlja.**
- **Gledajući spolja, automobil predstavlja jedinstven objekat.**

  Kada uđemo u njega, vidite da se on sastoji od više podsistema:
  - **za upravljanje,**
  - **za kočenje,**
  - **za signalizaciju,**
  - **za obezbeđivanje putnika,**
  - **za zagrevanje,**
  - **za komunikaciju itd.**

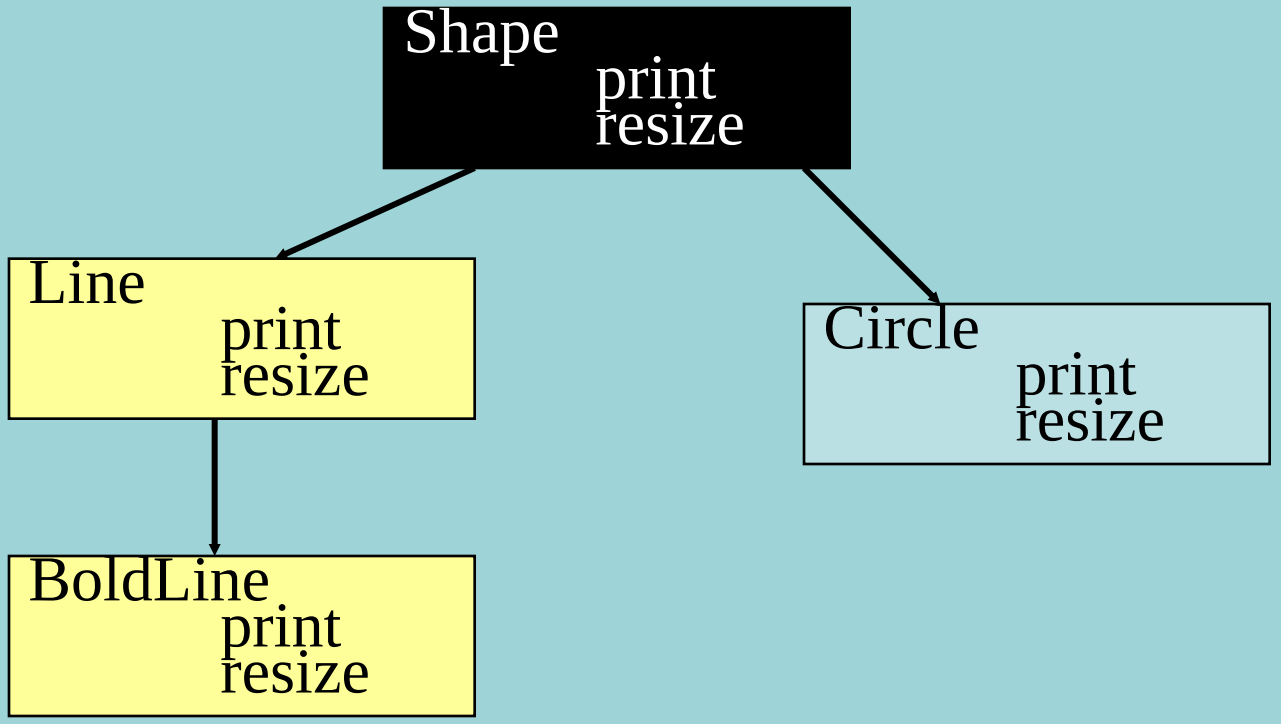
- **Svaki od ovih podsistema je sastavljen od delova koji su još više specijalizovani. Na primer, ozvučenje se sastoji od radio-aparata, CD plejera i/ili kasetofona. Važno je da shvatite da vi upravljate složenim sistemom automobila (ili bilo kojim drugim složenim sistemom) koristeći hijerarhijsko apstrahovanje.**

Apstrakcija

- **Hijerarhijsko apstrahovanje složenih sistema može se primeniti i na računarske programe.**
  - **Podaci iz tradicionalnih procesno orijentisanih programa apstrahovanjem se mogu pretvoriti u sastavne objekte.**
  - **Niz procesnih koraka može da postane zbirka poruka između dva objekta. Na taj način, svaki od ovih objekata opisuje sopstveno jedinstveno ponašanje.**
- **Ove objekte možemo smatrati posebnim celinama koje reaguju na poruke kojima im se saopštava da nešto urade. To je suština objektno orijentisanog programiranja.**

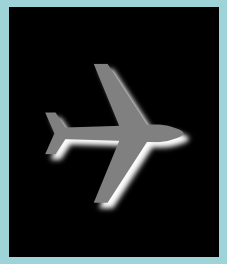
Apstraktne klase i interfejsi

– Osnovne klase koje služe kao osnova za generisanje novih klasa.



Apstrakcija

Mentalni proces izdvajanja nekih karakteristika isvojstava i isključivanja preostalih koja nisu relevantna



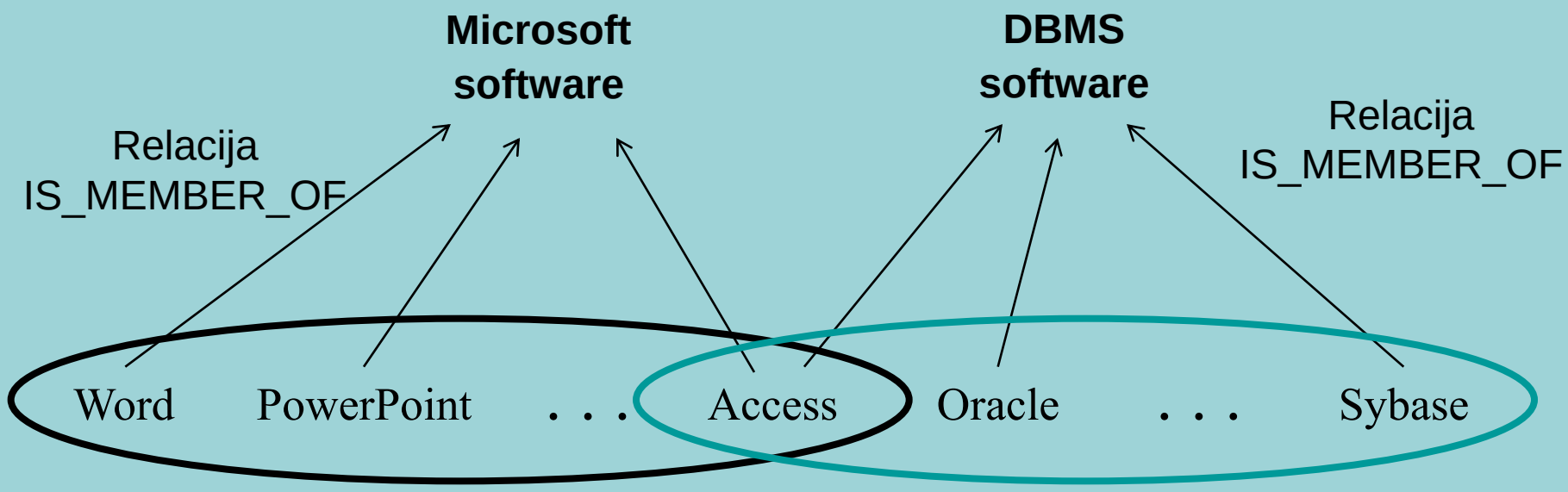
- ima krila, leti

- Apstrakcija je uvek sa nekim ciljem, namenom
- Istre stvari moguće je apstrakovati na više različitih načina
- Svaka apstrakcija je nepotpuna slika realnosti

➔ *Ne želimo potpunost, već samo adekvatno modeliranje!*

Tipovi apstrakcije

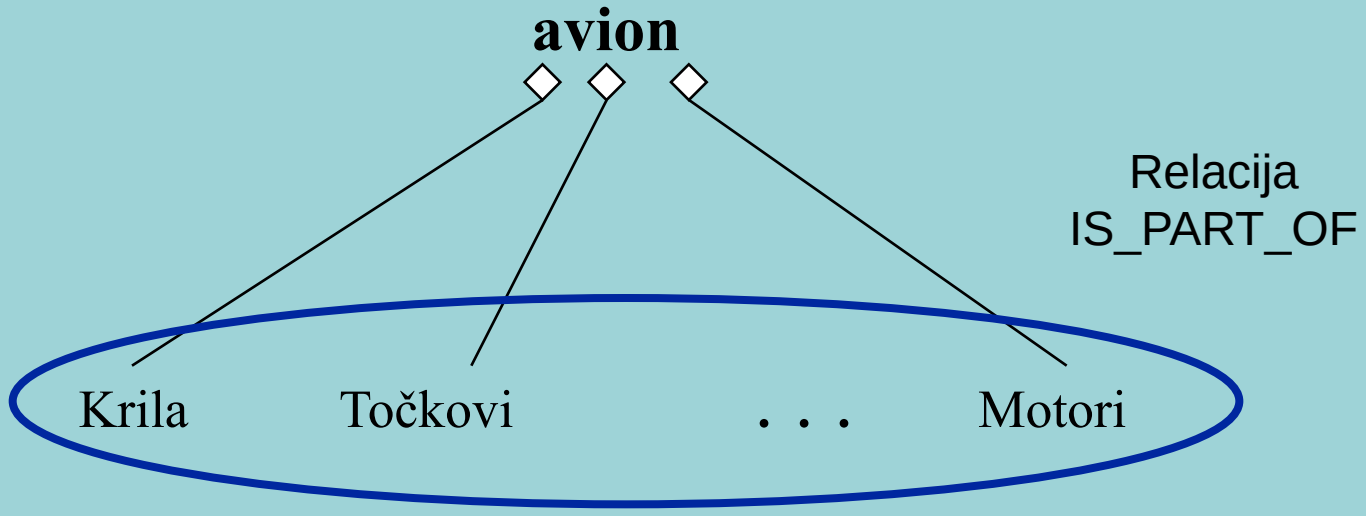
- Klasifikacija — grupa sličnih instanci objekta



➔ Ističu se zajednička svojstva i ignorišu posebna

Tipovi apstrakcije

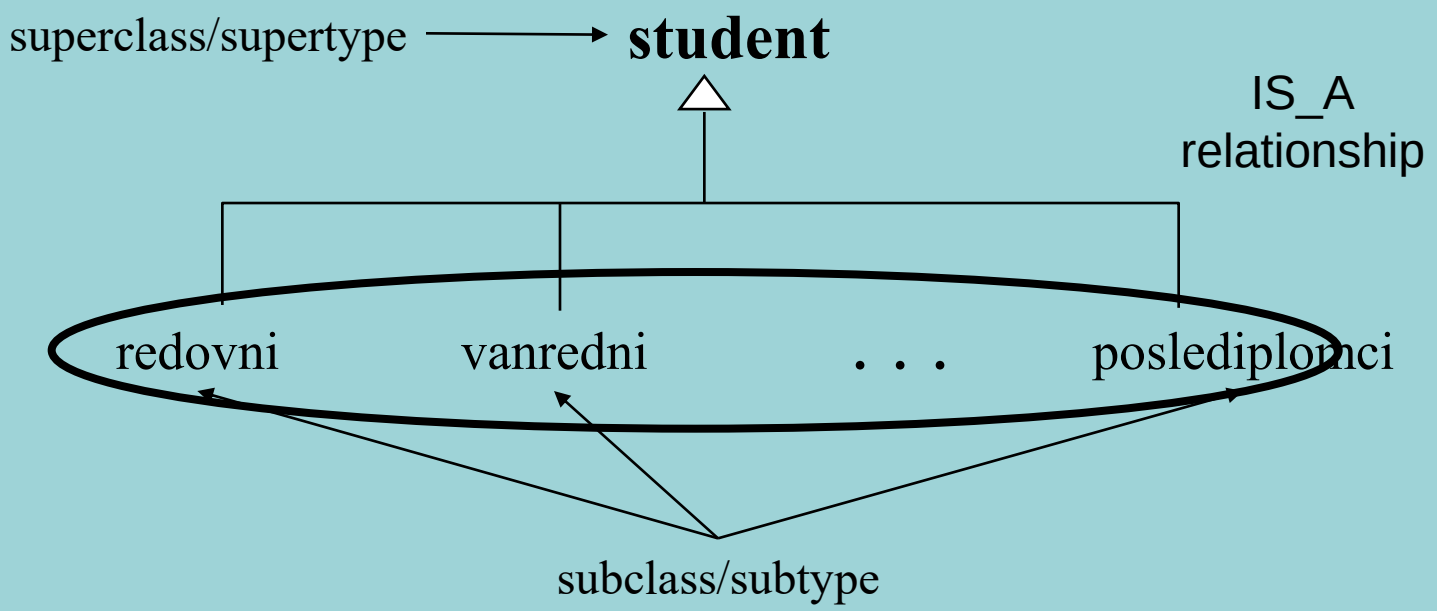
- agregacija — grupa različitih skupova objekata



➔ ignorišu se razlike između delova - Koncentrišemo se na činjenicu da oni čine celinu

Tipovi apstrakcije

- generalizacija — grupa sličnih skupova objekata

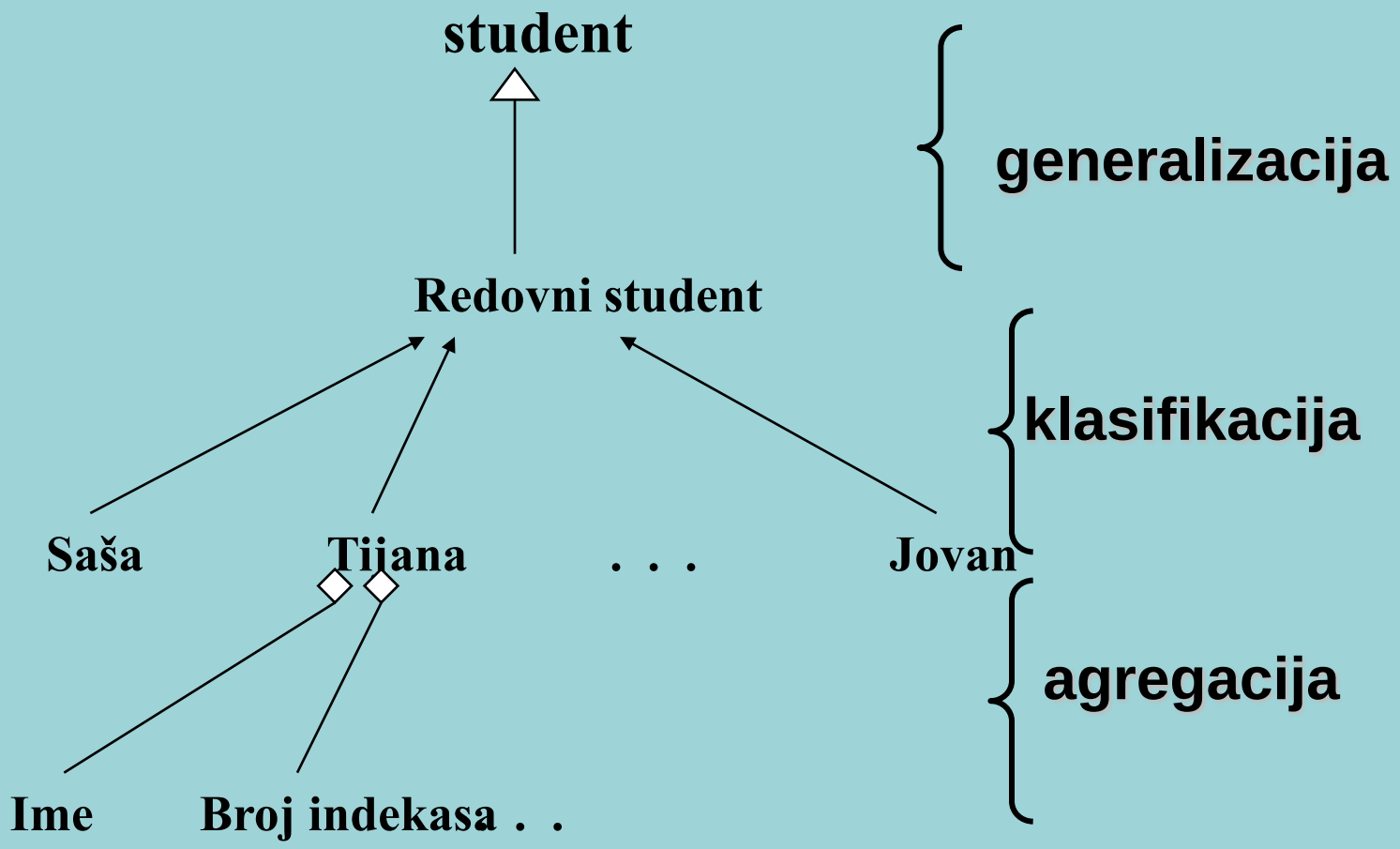


➔ Uočimo razliku između klasifikacije i generalizacije

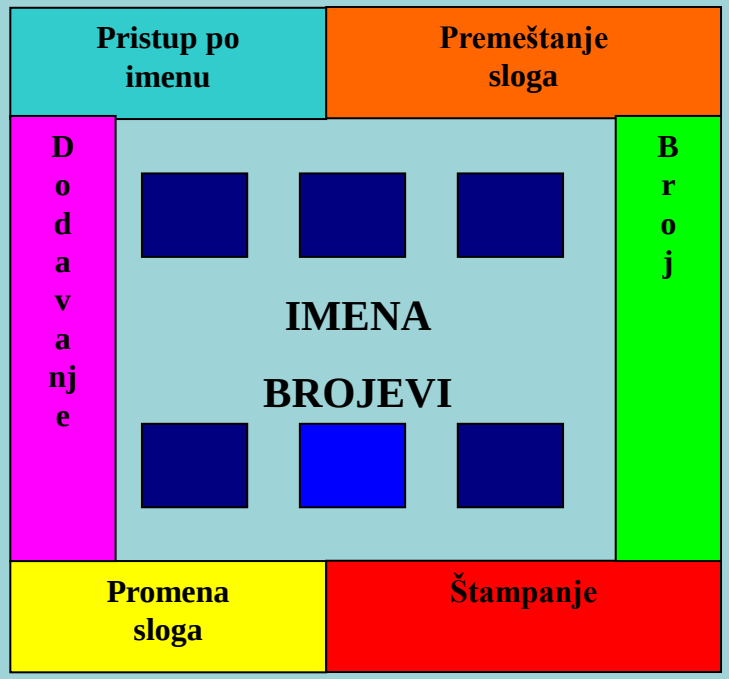
– klasifikacija– primenjuje se na individualne instance objekata

– generalizacija – na skup objekata (klase)

Tipovi apstrakcije



Apstraktni tipovi podataka - klase, objekti i poruke

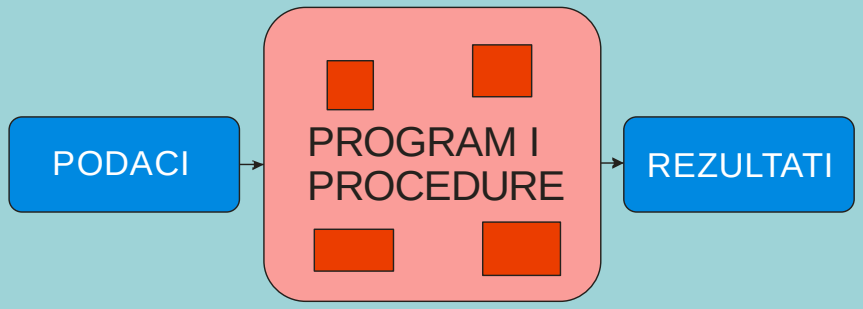


Imena korisnika i telefonski brojevi su podaci klase nad kojima se mogu izvršavati metodi klase kao što su: dodavanje novog broja, traženje po broju ili imenu i slično.

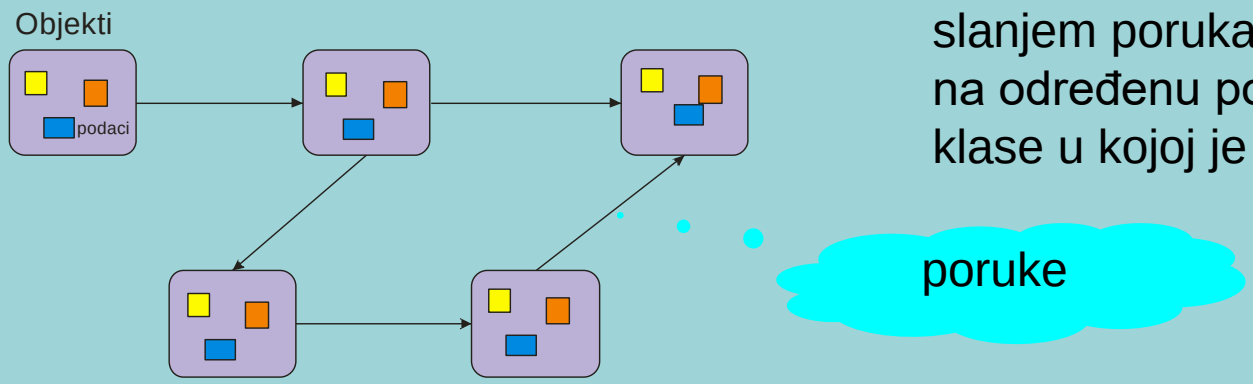
Klasa može da se posmatra kao šablon objekata *(template)* kojim se opisuje model po kome će se kreirati novi objekat. Klase se ponekad nazivaju fabrikama objekata, čime se naglašava algoritamska priroda kreiranja objekata.

Ključni pojam u terminologiji objektnih jezika su poruke. Kažemo da na objekte delujemo porukama, pri čemu svaka poruka predstavlja poziv metoda (funkcije ili procedure) definisanog u klasi kojoj objekat pripada. Objekat odgovara na poruku time što se izvršava odgovarajući metod

Apstraktni tipovi podataka - klase, objekti i poruke



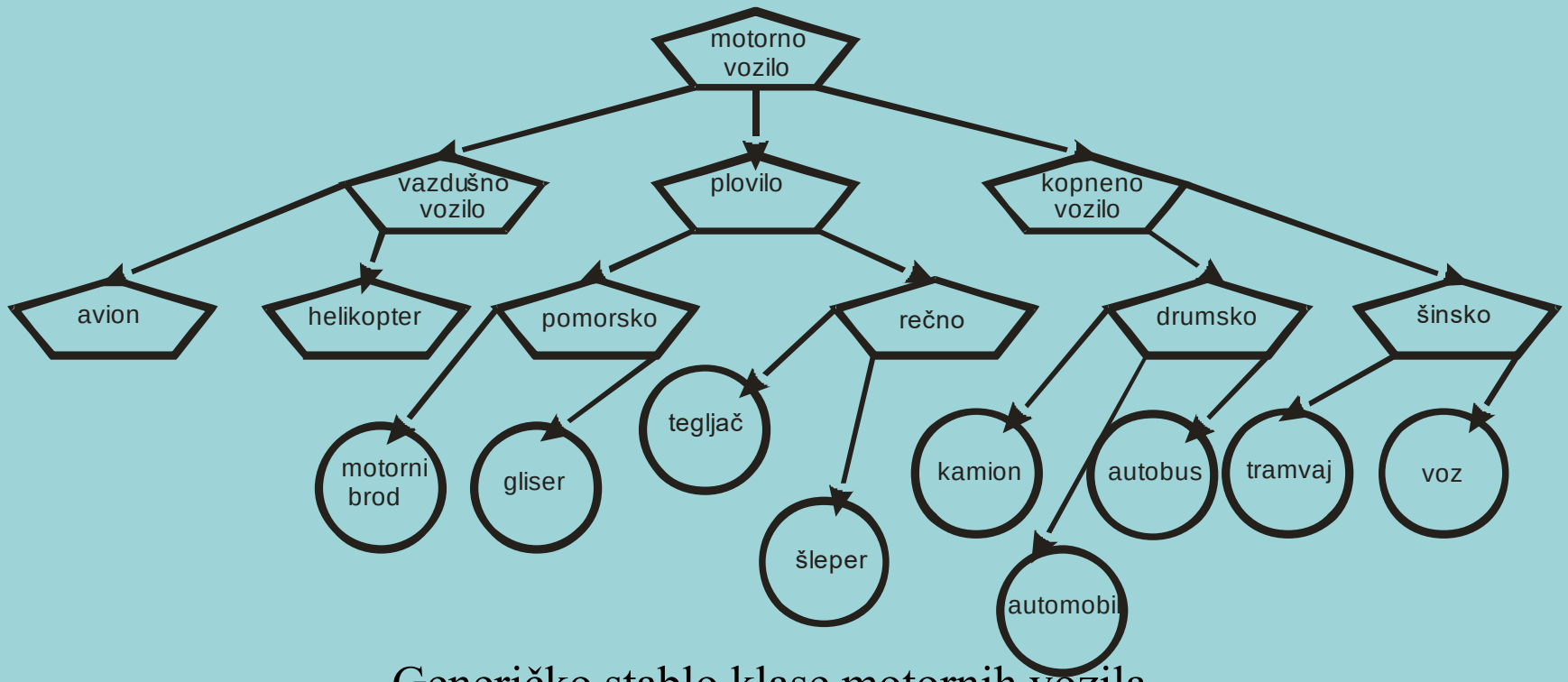
BLOK DIJAGRAM PROCEDURALNOG PROGRAMIRANJA



BLOK DIJAGRAM OBJEKTNOG PROGRAMIRANJA

Dok se kod proceduralnih jezika program može posmatrati kao skup potprograma (procedura i funkcija) kojima se prosleđuju podaci, kod objektnih jezika program se razvija kao skup objekata koji deluju jedni na druge slanjem poruka. Odgovor objekta na određenu poruku zavisi od klase u kojoj je definisan.

Apstraktni tipovi podataka - klase, objekti i poruke



Generičko stablo klase motornih vozila

Grupisanjem sličnih objekata stvara se klasa kao novi tip podataka koji sadrži zajednička svojstva objekata iz svog sastava.

Kapsuliranje

- **Kapsuliranje (engl. encapsulation) predstavlja mehanizam koji povezuje naredbe i podatke sa kojima one rade, i oboje štiti od spoljnih uplitanja i zloupotreba. Kapsulirani kôd dobija "zaštitnu čauru" koja štiti naredbe i podatke od proizvoljnog pristupa izvan čaure. Pristupanje naredbama i podacima unutar čaure strogo se kontroliše pomoću dobro definisanih standarda.**
  - **Da bismo ovu situaciju približili stvarnosti, razmotrimo ponašanje automatskog menjača u automobilu. Ovaj sistem kapsulira na stotine podataka o motoru, na primer, ubrzanje, zatim podatke o nagibu puta, kao i o položaju ručice menjača. Kao korisnik, vi možete da utičete na ovu složenu strukturu samo na jedan način: pomeranjem ručice menjača. Na ponašanje menjača ne možete da utičete pritiskajući žmigavce, na primer, ili pokrećući brisače. Shodno tome, ručica menjača jeste ono što predstavlja dobro definisan (u stvari, i jedini) način pristupa menjaču. Štaviše, ono što se događa unutar menjača ne utiče na objekte izvan njega.**

Kapsuliranje

- **Osnova kapsuliranja je klasa. Mada ćemo o klasama detaljnije govoriti u nastavku, potrebno je da na ovom mestu ukažemo na sledeće:**
  - **Klasa (engl. class) definiše strukturu i ponašanje (podatke i naredbe) zajedničke za skup objekata.**
  - **Svaki objekat određene klase ima strukturu i ponašanje definisane klasom, baš kao da predstavlja repliku klase.**
  - **Zbog ovoga se objekti ponekad nazivaju primercima ili instancama klase. Na taj način klasa predstavlja logičku konstrukciju; objekat predstavlja fizičku realnost.**

Kapsuliranje

- **Pošto je svrha klase da kapsulira složenost, postoje mehanizmi za skrivanje složenosti načina rada unutar klase. Svaka metoda ili promenljiva unutar klase može da bude privatna ili javna.**
  - **Javni (engl. public) deo klase predstavlja sve šta spoljni korisnici klase treba ili mogu da znaju.**
  - **Privatnim (engl. private) metodama i podacima može da pristupi samo kôd klase. Kôd izvan te klase ne može da pristupi privatnoj metodi ili promenljivoj.**
  - **Zaštićeni (engl. protected)**
- **Pošto privatne članove klase drugi članovi našeg programa mogu da dosegnu samo preko javnih metoda klase, na taj način se obezbeđujete od neodgovarajućih akcija.**

Nasleđivanje

- Većina ljudi smatra da je svet prirodno sačinjen od objekata međusobno povezanih na hijerarhijski način, npr. životinja, sisara i pasa. Ako bi želeli da životinje opišemo apstraktno, rekli bi da one imaju izvesne osobine, npr. veličinu, inteligenciju i vrstu skeletnog sistema. Životinje imaju i određene aspekte ponašanja: one jedu, dišu i spavaju. Ovakav opis osobina i ponašanja predstavlja definiciju klase životinja.

- Ako želimo da definišemo određeniju klasu životinja, npr. sisare, oni bi takođe morali imati određenije osobine, npr. mlečne žlezde i tip zuba. Kažemo da su sisari potklasa (engl. subclass) životinja, a životinje su sisarima natklasa (engl. superclass).
- Pošto su sisari samo uže definisane životinje, oni nasleđuju sve osobine životinja. ***Potklasa koja se nalazi duboko u hijerarhiji nasleđuje osobine svih svojih predaka u hijerarhiji klase.***

Nasleđivanje

- ***Nasleđivanje (engl. inheritance) je proces kojim jedan objekat dobija svojstva drugog objekta.***

- **Nasleđivanje je važno jer podržava koncepciju hijerarhijske klasifikacije (tj. odozgo nadole).**
  - **Na primer zlatni retriver je deo klase pas, koja je deo klase sisar, koja je deo veće klase životinja.**

- ***Bez postojanja hijerarhije, za svaki objekat bi se morale eksplicitno zadati sve njegove karakteristike.***

- **Međutim, ako postoji nasleđivanje, za objekat treba definisati samo one karakteristike koje ga čine jedinstvenim u klasi.**

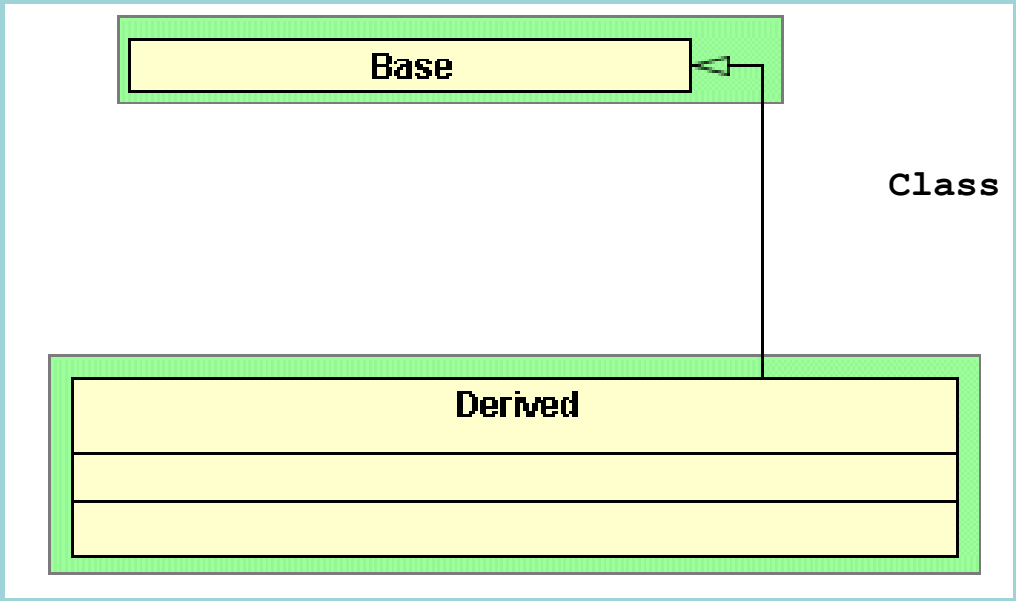
- **Opšta svojstva objekat može da nasledi od roditelja. Na taj način, nasleđivanje predstavlja mehanizam koji omogućuje da jedan objekat bude poseban slučaj nekog opštijeg objekta.**

Nasleđivanje

- **Nasleđivanje je povezano i sa kapsuliranjem.**
  - **Ako klasa kapsulira neke osobine, onda će svaka potklasa imati te osobine ali i osobine kojima se potklasa izdvaja.**
- **Ovo je ključna koncepcija koja omogućuje da složenost objektno orijentisanih programa raste linearnom, umesto geometrijskom progresijom.**
- **Nova potklasa nasleđuje sve osobine svih svojih predaka. Ona ne dolazi u dodir sa većinom ostalog koda u sistemu na nepredvidljiv način.**

Nasleđivanje

- **Nasleđuju se sva svojstva postojeće klase i njima se dodaju nova.**
- **Uspostavlja se “is-like-a” relacija**




```
Class Derived extends Base {}
```

Nasleđivanje

- Nasleđivanje omogućava da se definišu nove klase na osnovu postojećih.
- Nova klasa se definiše kao specijalizacija ili proširenje neke klase koja već postoji.
- Nova klasa inicijalno ima sva svojstva klase koju nasleđuje.
- Novoj klasi mogu da budu pridodata nova svojstva.
- Kada jedna klasa, u OO terminologiji podklasa ("subclass", "child class", "derived class"), nasleđuje neku klasu , u OO terminologiji superklasa ("superclass", "parent class", "base class"), onda podklasa nasleđuje sva svojstva superklase.

Nasleđivanje

- **Podklasi mogu biti pridodata nova svojstva (podaci, metodi, poruke), a neka svojstva (prvenstveno metodi) mogu biti izmenjena sa ciljem da se nova klasa prilagodi specifičnimpotrebama.**

- **Sva dodata svojstva i izmene odnose se na poklasu (subclas) dok originalna klasa (superclas) ostaje nepromenjena.**

- **Svojstva koja nisu definisana u podklasi automatski se preuzimaju iz njene superklase.**

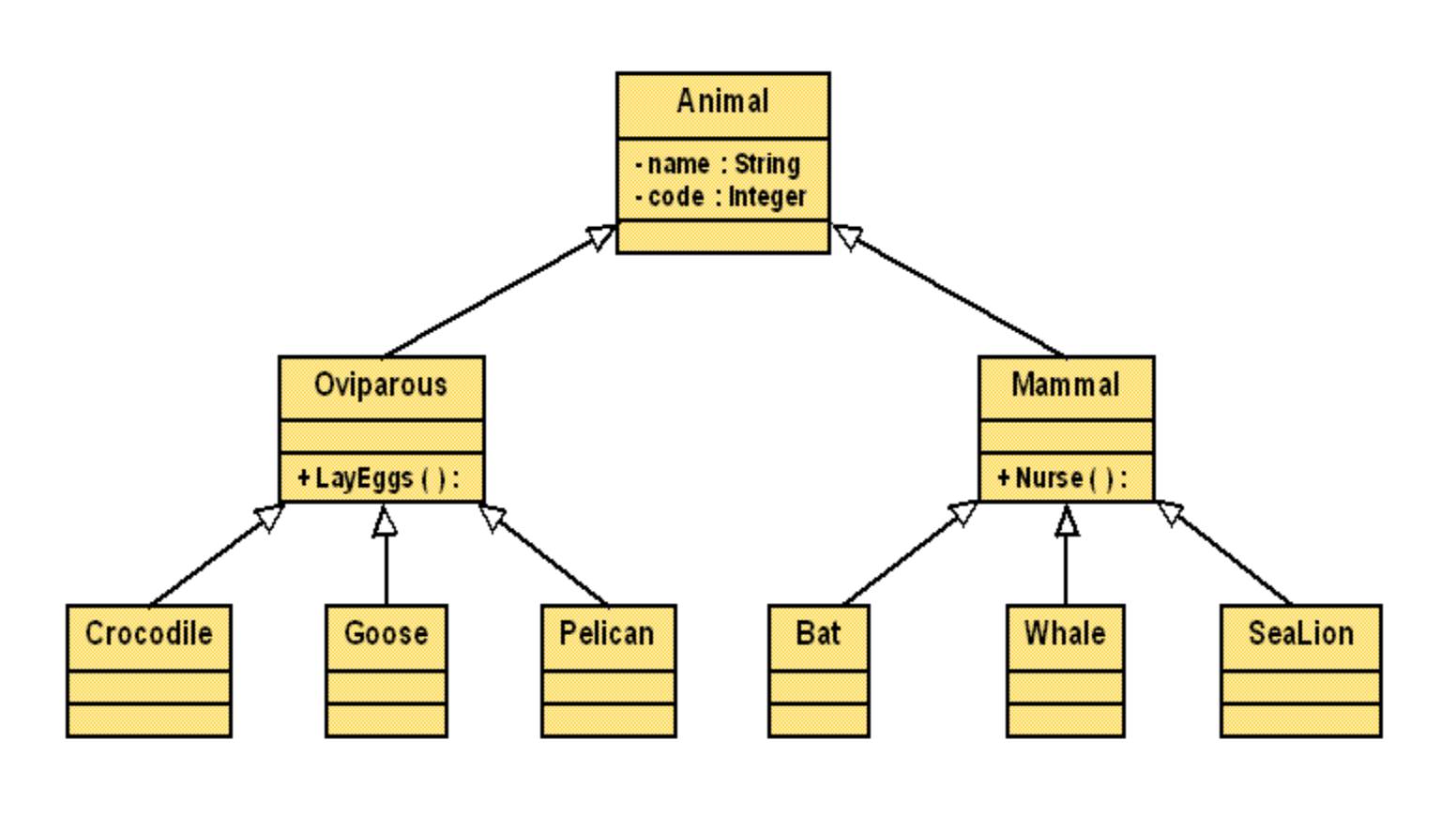
Šta se može menjati

- ***Nove poruke***, time se proširuje interfejs nove klase. Za svaku novu poruku treba da bude definisan odgovarajući metod.

- ***Novi podaci instanci,*** time se proširuje struktura objekata. Posmatrano sa strane apstrakcije objekti podklasa su specijalizovani (detaljnije opisani) od objekata superklasa. Subklase ne samo da imaju veću funkcionalnost već imaju i više svojstava..

- ***Postojeći metodi mogu biti predefinisani*****,** na taj način se modifikuje ponačanje objekta. U ovom slučaju govorimo o predefinisanju (***overriding)*** metoda.

Primer nasleđivanja

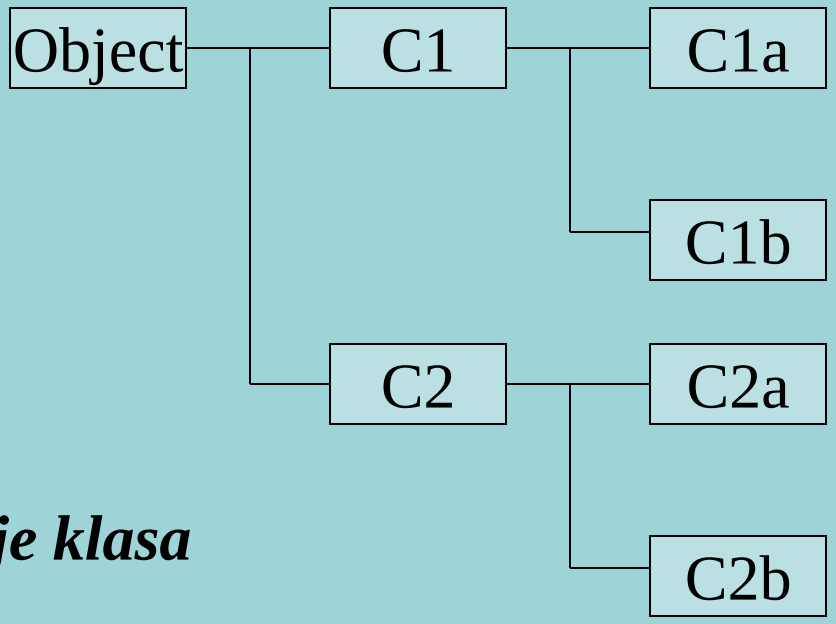


IS A relacija

- Nasleđivanjem se definiše IS A relacija između originalne klase i nove, izvedene klase.
- Kako objekti podklase mogu da prime sve poruke kao i objekti superklase oni se mogu koristiti svuda gde se mogu koristiti i objekti superklase.
- Svaki objekat jedne klase je i (is an) objekat njene superklase.

Hierarhija klasa

◆ Kada se nasleđivanje primeni višestruko dobija se struktura tipa stabla koja se obično naziva hierarhija klasa.



Primer hierarhije klasa

Polimorfizam

- **Polimorfizam (engl. polymorphism), reč grčkog porekla, znači "mnogo oblika" i predstavlja osobinu koja omogućuje da se *jedan način pristupa koristi za opštu klasu akcija.***
- **Specifičnost akcije biće određena tačnom prirodom situacije. Razmotrimo stek (strukturu "poslednji koji uđe, prvi izlazi"):**
  - **Možemo da imamo program kome su potrebne tri vrste steka: jedan za cele brojeve, drugi za brojeve u pokretnom zarezu, a treći za znake.**
- **Algoritam kojim se obrazuju stekovi uvek je isti, bez obzira na to što se u njima čuvaju različiti podaci. U jezicima koji nisu objektno orijentisani morali bi za svaki stek da napišemo poseban skup naredbi u kojima bi se koristila različita imena. Međutim, zbog postojanja polimorfizma, u C*++ možemo da definišemo opšti skup naredbi za stekove koji će imati ista imena.**

Polimorfizam

- **Uopšteno govoreći, koncept polimorfizma često se prikazuje frazom "jedan način pristupa, više metoda".**
  - ***Ovo znači mogućnost pravljenja opšteg načina pristupa za rad sa grupom srodnih aktivnosti.***

- **To pomaže smanjenju složenosti, omogućujući da se istim načinom pristupa zada opšta klasa akcija.**

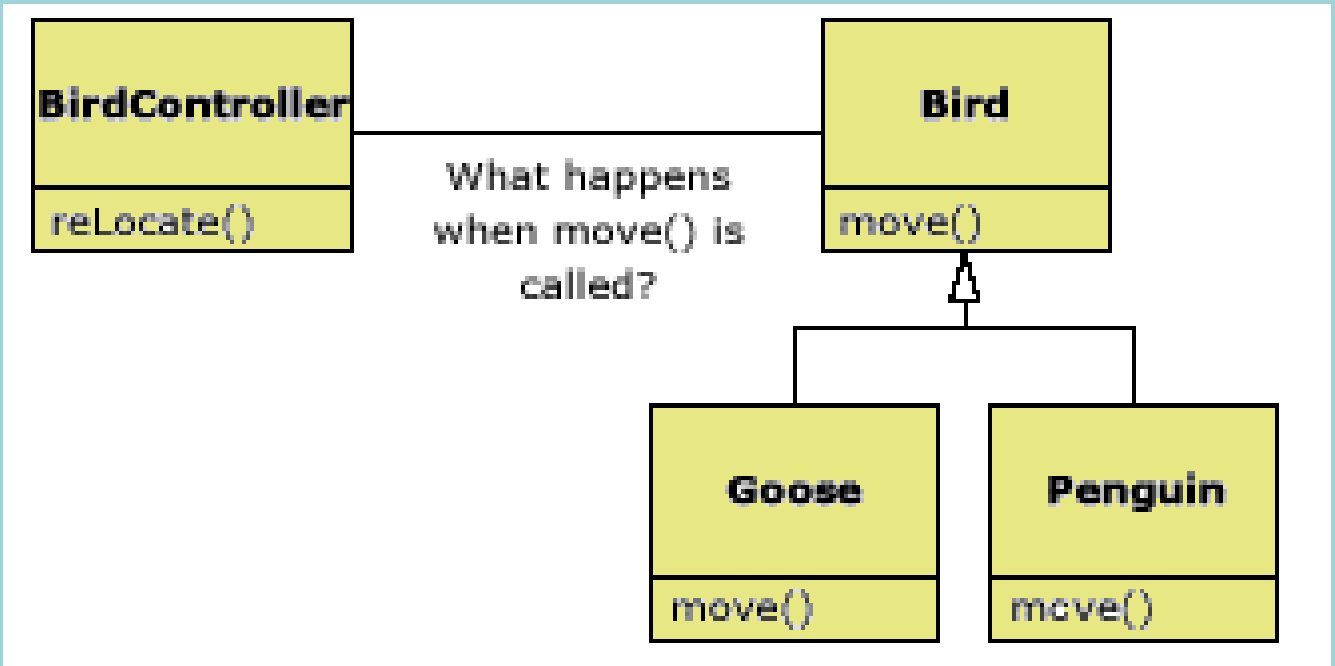
- **Na prevodiocu je da odabere specifičnu akciju (tj. metodu) shodno situaciji. Programer ne mora ručno da bira. Potrebno je samo da zapamti i iskoristi opšti pristup.**

Polimorfizam

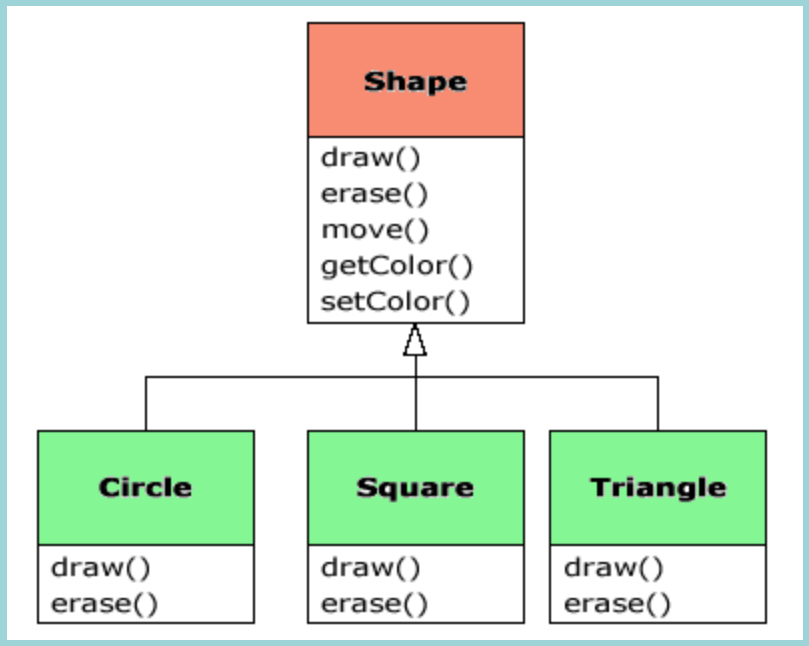
- Polimorfizam se često navodi kao najače svojstvo OO jezika.
- Termin označava “više formi” - "many forms", i u kontekstu objektnih jezika ukazuje na mogućnost da se metodi više objekata pozivaju preko istog interfejsa.
- Sposobnost da raziličiti objekti odgovore ne iste objekte (na sebi svojstven način).
- Koristi se isti interfejs za ratličite objekte.

Polimorfizam

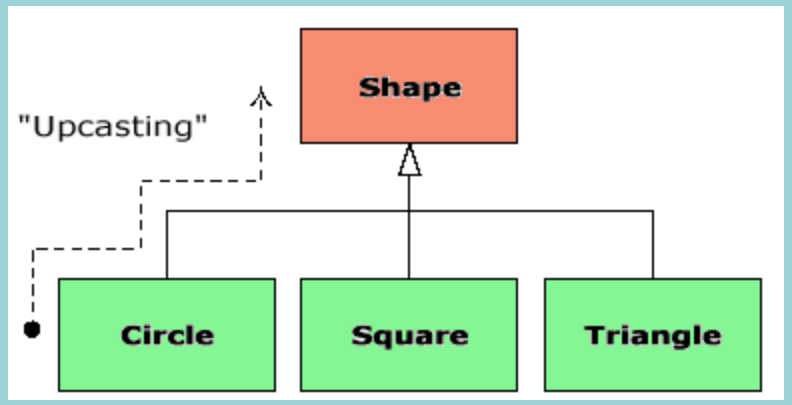
- Zakasnelo povezivanje.



Još jedan primer polimorfizma




```
Shape c = new Circle();
Shape t = new Triangle();
Shape s = new Square();
// ...
c.draw();
t.draw();
s.draw();
```



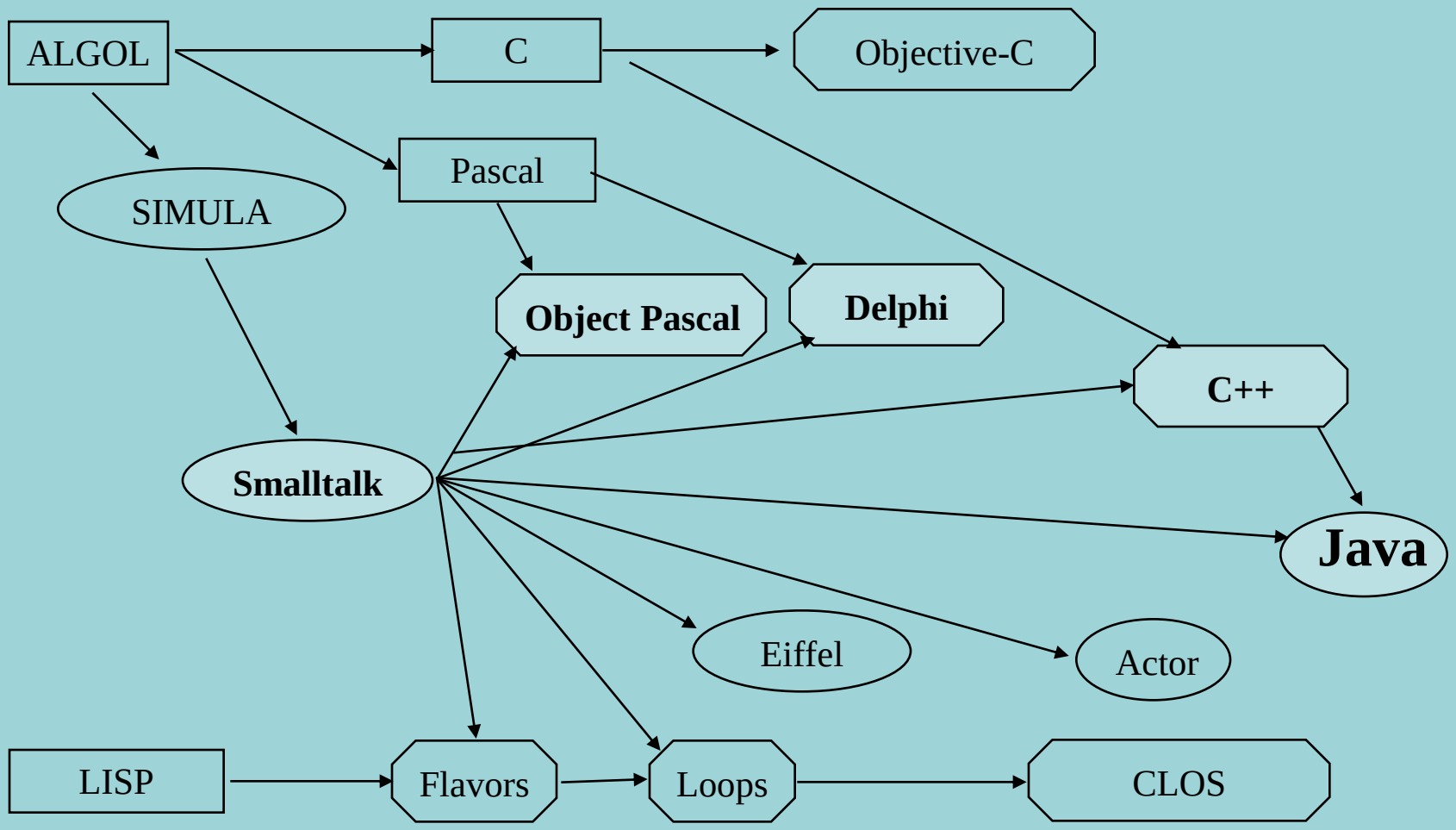
Polimorfizam, kapsuliranje i nasleđivanje međusobno sarađuju

- **Kada se primene na odgovarajući način, polimorfizam, kapsuliranje i nasleđivanje stvaraju programsko okruženje koje podržava razvoj mnogo robusnijih i prilagodljivijih programa nego što omogućuje procesno orijentisan model.**
- **Dobro dizajnirana hijerarhija klasa predstavlja osnovu za ponovljeno korišćenje koda u čije smo razvijanje i proveravanje uložili vreme i trud.**
- **Kapsuliranje omogućuje da tokom vremena menjamo unutrašnju konstrukciju ne remeteći kôd koji se oslanja na javni interfejs naših klasa.**
- **Polimorfizam omogućuje da napravimo čist, smislen, čitljiv i elastičan kôd.**

Polimorfizam, kapsuliranje i nasleđivanje međusobno sarađuju

- Od dva primera iz stvarnog sveta koja smo koristili, automobil potpunije ilustruje snagu objektno orijentisanog dizajna. O psima je zabavno raspravljati kada govorimo o precima, ali automobil više liči na program.
  - Svi vozači se oslanjaju na nasleđivanje kada sedaju u različite vrste (potklase) automobila.
  - Nije važno da li je u pitanju školski autobus, Mercedes sedan, Porše ili porodični kombi, svi vozači će uglavnom pronaći i umeti da koriste upravljač, kočnice i pedalu gasa.
  - Nakon izvesnog krčanja zupčanika, većina vozača će shvatiti razliku između običnog i automatskog menjača, jer svi u osnovi shvataju njihovu natklasu, prenosni mehanizam.

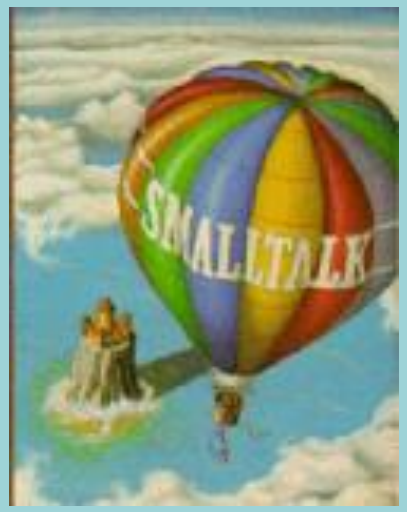
OO jezici


ALGOL
C
Objective-C
SIMULA
Pascal
Object Pascal
Delphi
C++
Smalltalk
Java
Eiffel
Actor
LISP
Flavors
Loops
CLOS

Smalltalk

- **Smalltalk-80 verzija iz 1980.**
  - Program se sastoji od poruka čija sintaksa odgovara uprošćenom engleskom jeziku
  - Na objekte se deluje porukama koje prestavljaju pozive metoda.
  - Dinamička implementacija, kasno povezivanje.

```
krug := Circle new;
krug center: x @ y radius:r;
vert := Line new; vert start: x @ (y-r);
vert end: x @ (y+r);
hori := Line new; hori start: (x-r) @ y;
hori end: (x+r) @ y;
ckrug display; vert display; hori
display;
```



C++

- **1980 Bjarne Stroustrup je projektovao skup jezika koji su bili nazvani "C With Classes" – što je poslužilo kao osnova za definisanje jezika C++.**
  - **Proširenje programskog jezika C**
  - **Statička implementacija. Klase su ujedno i tipovi podataka**

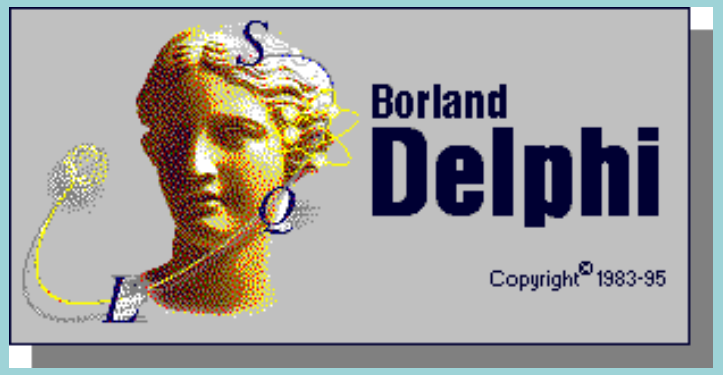
```
krug = new Circle(x,y,r)
vert = new Line(x, x-r, x, y+r)
hori = new Line(x-r, y, x+r, y)
krug -> display()
vert -> display()
hori -> display()
```

Object Pascal - Delphi

- **In 1986 Apple realizuje Object Pascal za Mac.**

- **1995 pojavljuje se Delphi- popularna verzija razvojnog okruženja (RAD tool) zasnovanog na Object Pascalu.**

```
New(krug);
krug.ICircle(x,,y,,r);
New(vert); vert.ILine(x,x-
r,x,y+r);
New(hori); hori.ICircle(x-
r,y,x+r,y);
krug.display; vert.display,
hory.display
```



Java

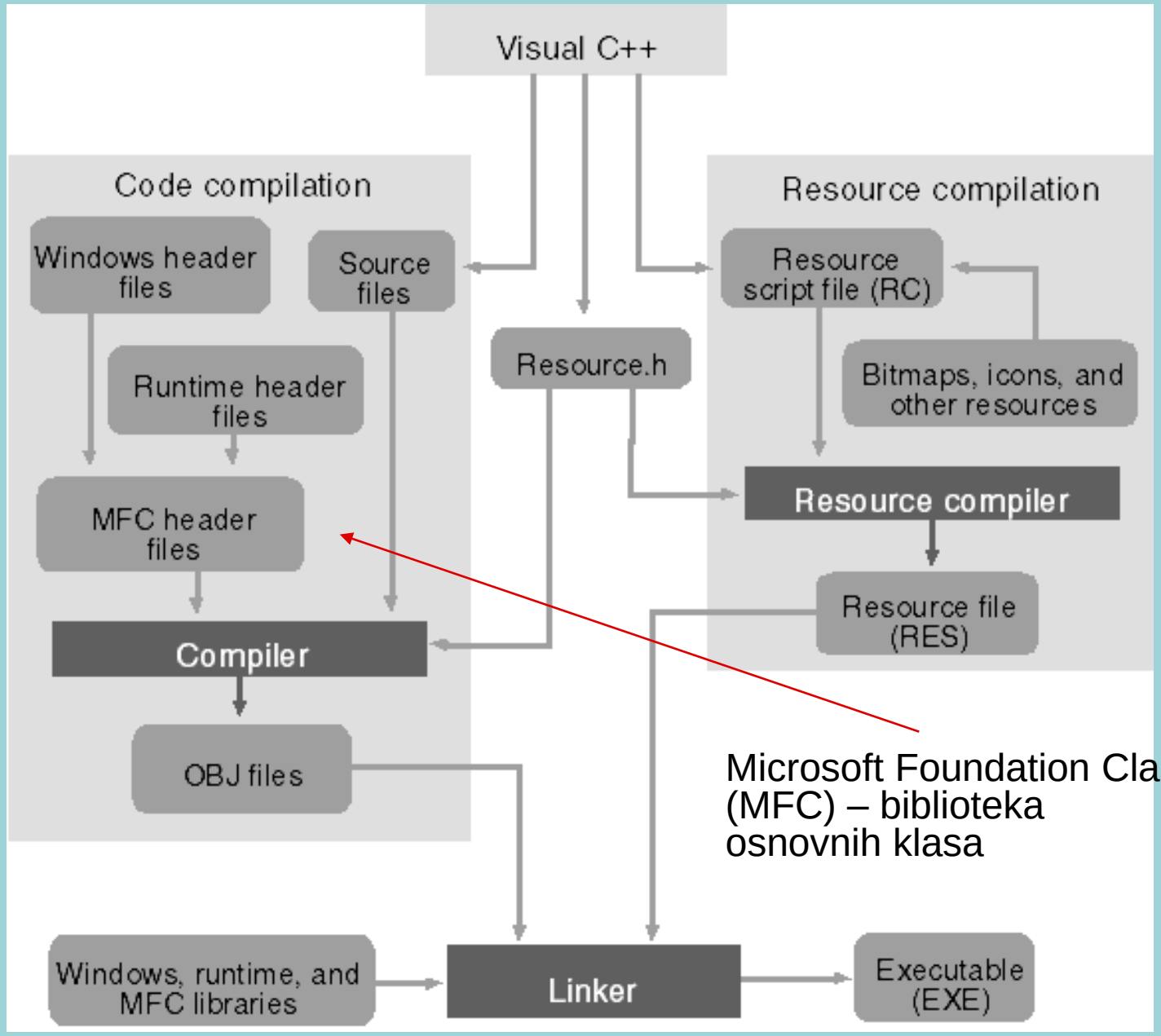
- **May 1995.**
- **Popularan je jer je jednostavniji od C++.**
- **Ima primenu u razvoju Internet aplikacija**

O programskom jeziku *C++*

- *ANSI* stnadard za jezik *C++* usvojen je l4. II. 1997. godine. Jezik *C ++* danas je jedan od najmoćnijih jezika za objektno orijentisano programiranje. Operativni *sistem UNIX* predstavlja prirodno okruženje za jezik *C++*, kao što je to slučaj i sa jezikom C. Ali, s obzirom da se radi o jeziku opšte namene, nudi se i pod drugim operativnim sistemima kao što je *MS-DOS* i *MS-Windows* na danas vrlo populamirn ličnim računarima *tipa IBM-PC*.

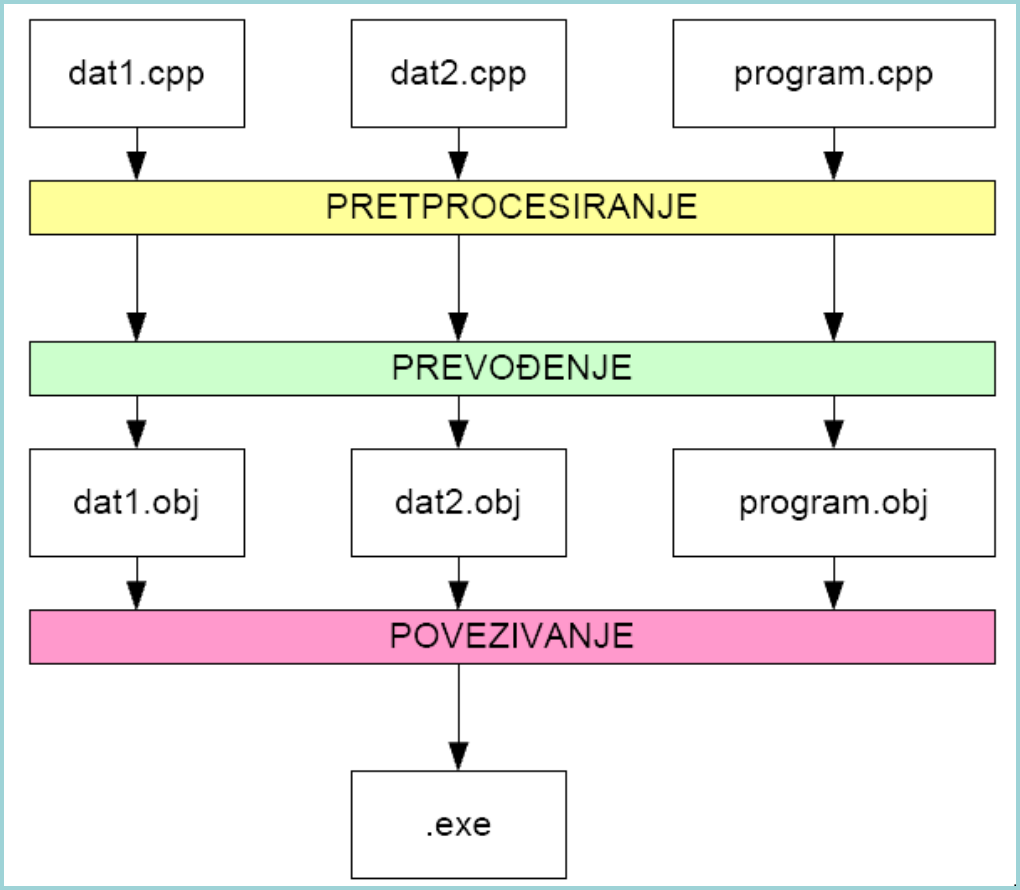
O programskom jeziku *C++*

- Ansi standard je pokušaj da se osigura da je C++ portabilan, da će se programski kod koji pišete za Microsoftov prevodilac, prevesti bez grešaka i kada koristite prevodilac nekog drugog proizvođača. Pošto programski kod odgovara ANSI standardu trebalo bi da se prevede bez greške na Macintosh, Windows ili Alpha računarima. Za većinu ljudi koji uče C ++ ANSI standard će biti neprimetan.
- Standard je stabilan već neko vreme i svi glavni proizvodači ga podržavaju. Učinili smo maksimalne napore da budemo sigurni da sav programski kod u ovom izdanju knjige odgovara ANSI standardu.



Visual C++ application build process

Visual C++ application build process



Pre nego što prevodilac počne sa prevonenjem izvornih datoteka dat1.cpp, dat2.cpp i program.cpp pretprocesor će izvršiti odgovarajuće pripreme teksta, kao što je umetanje odgovarajućeg zaglavlja zaglavlje.hpp, a preko njega i standardno zaglavlje <iostream>. Zatim će prevodilac (kompajler) prevesti izvorne datoteke u nastaće datoteke dat1.obj, dat2.obj i program.obj iz kojih će se povezivanjem (linkovanjem) dobiti izvršna datoteka. Ceo tok je ilustrovan na slici:

Ulaz - Izlaz

- Ulaz i izlaz podataka treba da omogućava komunikaciju programa sa spoljnim svetom radi unošenja podataka za obradu i prikazivanje ili odlaganje rezultata.
- Zbog toga ne može da se zamisli program bez ulaza i izlaza podataka. Uprkos tome, ulaz i izlaz podataka nije deo jezika *C++* u tom smislu što ne postoje zasebne naredbe za izvodenje tih radnji.
- Umesto toga postoje odgovarajuće bibliotečke klase za njihovo ostvarivanje.

Ulaz - Izlaz

- U programskom jeziku C++ dodeljena su nova značenja operatorima << i >>. Ukoliko je prvi operand referenca na tekstualnu datoteku onda te operatore koristimo za ulaz/izlaz podataka.

- Potrebne deklaracije za primenu operatora << i >> se nalaze u zaglavlju <iostream> (o čemu će kasnije biti više reči), pa će zbog toga ubuduće naši programi počinjati sa:

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

Za prelazak u novi red koristi se manipulator endl (end line), ali isto tako može da se koristi i karakter \n.

Ulaz - Izlaz

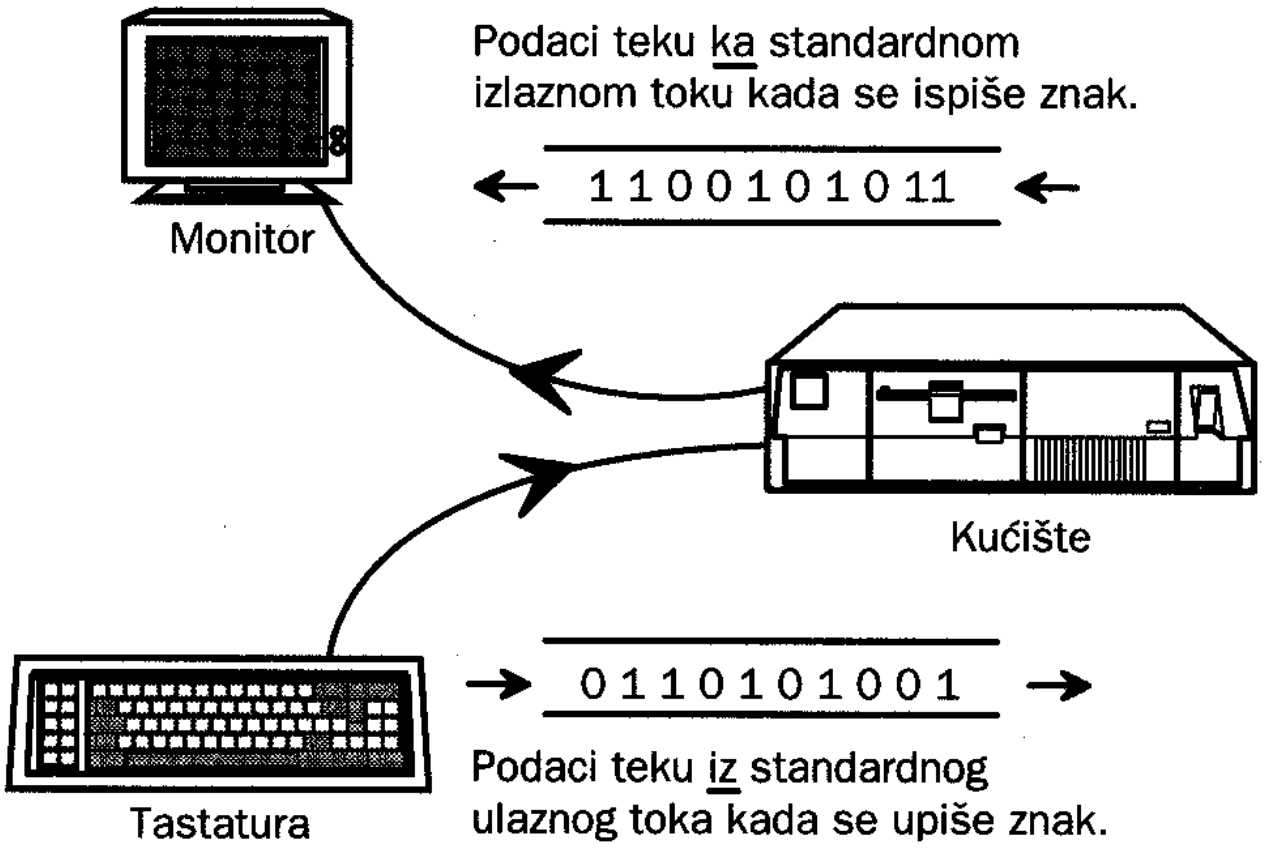
- *Za rad sa datotekama* (tokovima na magnetnim diskovima) postoje tri klase opisane u zaglavlju fstream.h.:
  - Klasa ofstream predviđena je samo za izlazne datoteke,
  - klasa ifstream samo za ulazne datoteke, dok
  - klasa fstream za ulazno-izlazne datoteke.

Ulaz - Izlaz

- Za rad sa tokovima u operativnoj memoriji, takođe postoje tri klase. Opisane su u zaglavlju strstream.h.

  – Klasa ostrstream služi samo za smeštanje podataka u tokove,
  – klasa istrstream samo za uzimanje podatka iz tokova, dok
  – klasa strstream omogućava kako uzimanje tako i smeštanje podatka u tokove.

Ulaz - Izlaz


Podaci teku ka standardnom
izlaznom toku kada se ispiše znak.
cout<<str
1 1 0 0 1 0 1 0 11
Monitor
Kućište
0 1 1 0 1 0 1 0 0 1
Tastatura
Podaci teku iz standardnog
ulaznog toka kada se upiše znak.
cin>>str

Ulaz - Izlaz

- **cin** -glavni (standradni) ulaz tipa istream. Predstavlja tastaturu dok se ne izvrši skretanje glavnog ulaza unutar samog programa ili u komandi operativnog sistema za izvršavanje programa.
- **cout** -glavni (standardni) izlaz tipa ostream. Predstavlja ekran monitora dok se ne izvrši skretanje glavnog izlaza unutar samog programa ili u komandi operativnog sistema za izvršavanje programa. Koristi se za ispisivanje podataka koji čine rezultate izvršavanog programa.

Ulaz - Izlaz

- **Letimičan pogled na operatore toka :**

  << i >>

- Operatori toka << **i** >> alternativa su funkcijama printf, scanf i ostalima iz datoteke stdio.h.
- Ovi operatori imaju dve prednosti:

  – ne moramo da koristimo oznake formata ako smo zadovoljni podrazumevanim formatom;
  – značenje operatora se možete proširiti tako da rade i sa vašim klasama.

Ulaz - Izlaz

Evo jednostavnog programa, koji čita dva broja u formatu pokretnog zareza i ispisuje njihov zbir:

f.ja stdio.h

```
#include <stdio.h>
void main()
{
double a,b;
printf("Unesite prvi broj: ");
scanf("%lf", &a);
printf("Unesite drugi broj: ");
scanf("%lf", &b);
printf("Zbir je %lf\n", a + b);
}
```

f.ja iostream

```
#include <iostream>
using namespace std;
void main()
{
double a,b;
cout << "Unesite prvi broj: ";
cin >> a;
cout << "Unesite drugi broj: ";
cin >> b;
cout << "Zbir je ";
cout << a + b << endl ;
}
```

Ulaz - Izlaz

Uočite sledeće važne tačke u verziji koja koristi operatore toka cin i cout:

- **Koristi se datoteka zaglavlja iostream.h, a ne datoteka stdio.h.**
- **Nisu potrebne oznake formata; način prenosa određuje tip objekta (u ovom primeru a i b).**
- **U tom pogledu, operatori toka su nalik iskazu PRINT u BASIC-u i malo se lakše koriste nego funkcije printf i scanf.**
- **Operator adrese (&) ne primenjuje se na operande kad se koristi cin, kao što je slučaj kod funkcije scanf. (Tu ulazni tokovi više liče na BASIC, jer koriste reference argumenata.)**

Podaci teku ka standardnom izlaznom toku (cout), što je obično ekran:

cout << "Unesite prvi broj: ";

- Podaci teku iz standardnog ulaznog toka (**cin**) , što je obično tastatura:

cin >> a;

Ulaz - Izlaz

U predhodnom primeru, poslednja dva reda :

- ***cout << "Zbir je ";***
- ***cout << a + b ;***

  mogu se napisati i zajedno:
- ***cout << "Zbir je " << a + b ;***

Operatori pomeranja su asocijativni sleva nadesno, što znači da će prvo biti izračunat sledeći izraz: cout « "Zbir je " .Kao što je u jezicima C i C++ uobičajeno, ovaj izraz radi dve stvari: šalje znakovni niz toku cout (što bi se moglo nazvati sporednim efektom), a zatim se izračunava i vraća vrednost. Vrednost izraza u obliku cout << argument jeste sam tok cout. Zbog toga se cout << " Zbir je " zamenjuje tokom cout.

Zbog toga se izraz izračunava na sledeći način:

- ***cout <<"Zbir je " <<a + b;***
- ***(cout <<"Zbir je “) << a + b; //ispiši znakovni niz.***
- ***cout << a + b; // ispiši a + b***

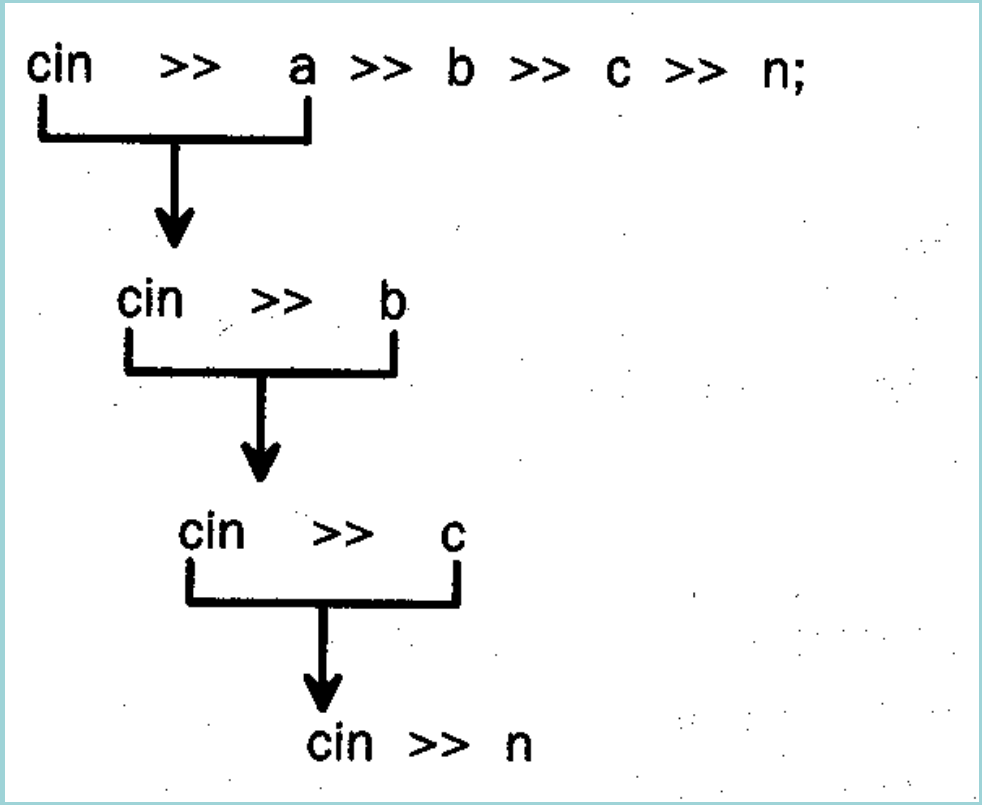
To je trik u C++-u koji omogućava da u velikom, složenom izrazu uvek iznova koristimo objekat cout. U ovom slučaju se prvo ispisuje znakovni niz, a zatim a + b.

Ulaz - Izlaz

Napisati program koji izračunava i ispisuje zbir dva unešena broja.

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

int main() {
    int x, y;
    cout<<"Ovo je program za racunanje zbira dva broja."<<endl;
    cout<<"Unesite brojeve... "<<endl;
    cin>>x>>y;
    cout<<"Broj x je: "<<x<<endl;
    cout<<"Broj y je: "<<y<<endl;
    cout<<"Njihov zbir je: "<<x+y<<endl;
    return 0;
}
```

Ulaz - Izlaz



Ovaj trik možete upotrebiti i za čitanje nekoliko brojeva u istom iskazu, jer svaki izraz oblika cin >> argument daje rezultat cin:

cin >> a >> b >> c >> n;

Ulaz - Izlaz

- *Operator za ulaz* - *cin* >> (dvostruko veće – prevodi se kao jedan znak)

  sintaksa je: cin >> <promenljiva>

- gde je cin objekat (promenljiva) klase ili tipa istream, unet sa tastature
- semantika je:

cin >> <promenljiva>;

cin >> <promenljiva> {>><promenljiva>};

cin >> promenljiva1>> promenljiva2>> ...>>promenljivaN;

Ulaz - Izlaz

- Da bi podatke uneli sa tastature moramo ih ukucati i preneti u operativnu memoriju pritiskanjem tastera ENTER kao što sledi na primeru:
- double a,b,c;
- cin >> a>>b>>c;
- može biti uneto na razne načine pretpostavljajući da je a=1.1,b=2.2,c=3.3 :

- **1.1 2.2 3.3 ENTER**
- **1.1 2.2 ENTER 3.3 ENTER**
- **1.1 ENTER 2.2 3.3 ENTER**
- **1.1ENTER 2.2ENTER 3.3 ENTER**

Ulaz - Izlaz

- *Operator za izlaz* - *cout* << (dvostruko manje - prevodi se kao jedan znak)

  sintaksa je: ***cout << <izraz>***

- gde je cout objekat (promenljiva) klase ili tipa ostream, unet sa tastature
- semantika je:

cout<< <izraz>;

cout<< <izraz> {<< <izraz>};

cout<< izraz1<<izraz2<<...<<izrazN;

Ulaz - Izlaz

- Šta biva ako broj želite da ispišete u heksadecimalnom ili oktalnom formatu? Funkcijom printf možete da uradite sledeće:

int n = 16;
printf("n je %x heksadecimalno, %o oktalno, i %d dekadno.\n", n, n, n) ;

- Rezultat ovoga primera je:
- n je 10 heksadecimalno, 20 oktalno i 16 dekadno.

Sledeći primer koristi operatore toka i ispisuje isti rezultat:

- ***int n = 16;***
- ***cout << "n je " << hex << n <<" heksadecimalno, ";***
- ***cout << oct << n << " oktalno, i ";***
- ***cout << dec << n << " dekadno." << endl ;***

FUNKCIJE STANDARDNIH BIBLIOTEKA

- Na ovom mestu nije loše napomenuti da standard ISO 98 jezika C++ predviđa da standardna biblioteka jezika C++ obavezno mora sadržati biblioteke sa sledećim zaglavljima:

algorithm	**bitset**	**cassert**	**cctype**	**cerrno**	**cfloat**
ciso64	**climits**	**clocale**	**math**	**complex**	**csetjmp**
csignal	**cstdarg**	**cstddef**	**cstdio**	**cstdlib**	**string**
ctime cwchar	**cwtype**	**deque**	**exception**	**fstream**	
functional	**iomanip**	**ios**	**iosfwd**	**iostream**	
iterator	**limits**	**list**	**locale**	**map**	**memory**
new	**numeric**	**ostream**	**queue**	**set**	
stack	**stdexcept**	**streambuf**		**string**	**typeinfo**
utility	**valarray**		**vector**		

- skup *nestandardnih biblioteka:* biblioteka sa zaglavljem “Windows.h“ koja služi za pisanje Windows aplikacija

FUNKCIJE STANDARDNIH BIBLIOTEKA

- **Korišćenje direktive – Namespaces**

- Biblioteka **standard library** je prošireni set rutina koje su napisane da bi mogli izvršavati različite zadatke: naprimer, rad sa input i output sistemima, izvršavanje osnovnih matematičkih operacija, itd. Umesto da sami pišemo te rutine, jednostavno ih možemo pozvati iz standardnih biblioteka i koristiti ih u našem kodu. Fajl iostream je samo jedan deo header fajlova koji sadrži rutine standardne biblioteke. (Možemo videti ceo set fajlova standardne biblioteke u MSDN online help fajlovima u **Visual C++ Documentation \ Reference \ C/C++ Language and C++ Libraries \ Standard C++ Library Reference**.)

- Svi entiteti C ++ biblioteka su proglašeni ili definisani u jednom ili više standardnih zaglavlja - headera. Da bi koristili entitete biblioteke u programu, napisaćemo i inkludovati direktivu koja relevantno definiše standardno zaglavlje. Standardna C ++ biblioteka sastoji se od 50 potrebnih zaglavlja. Ova implementacija takođe uključuje dva dodatna zaglavlja , < hash_map > i < hash_set > , koje se ne traži od C ++ Standarda, sa ukupno 52 zaglavlja . Ove 52 biblioteke C ++ zaglavlja (sa dodatnih 18 Standard C zaglavlja) predstavljaju osnovnu implementaciju C ++ biblioteka.

FUNKCIJE STANDARDNIH BIBLIOTEKA

- Kodovi svih rutina standardnih biblioteka sadržani su u naredbi **namespace** std. Tako, svaka standardna rutina pripada opciji namespace std. Svaki header fajl standardne biblioteke prilaže nekoliko rutina pozivom namespace std.

- Kod našeg programa ne pripada funkciji namespace std. Prema tome, prilikom korišćenja izlaznih rutina, (naprimer iostream header fajlova), moramo reći računaru da ove rutine pripadaju direktivi namespace std. Koristićemo:

  - **using namespace std;**

- Ovom linijom u našem programu, kompajler zna da ćemo koristiti rutine koje pripadaju standardnoj biblioteci.

- Međutim, u jeziku C++ možemo normalno raditi i sa već poznatim direktivama za korišćenje ulaza/izlaza, ali je ne možemo koristiti ako koristimo prethodnu!!!

  - **#include<iostream.h>**

FUNKCIJE STANDARDNIH BIBLIOTEKA

Функције стандардних библиотека

Математичке функције из библиотеке math.h

- fabs апсолутна вредност
- log логаритам
- sqrt квадратни корен
- pow степеновање
- ...

FUNKCIJE STANDARDNIH BIBLIOTEKA

Функције стандардних библиотека

Стринг функције из библиотеке string.h

- strlen — дужина стринга
- strcmp — поређење стрингова
- strcpy — копирање једног стринга од почетка другог
- strcat — копирање једног стринга у продужетку постојећег садржаја другог
- ...

FUNKCIJE STANDARDNIH BIBLIOTEKA

Функције стандардних библиотека

Стринг функције из библиотеке stdlib.h

- rand — псеудо-случајно генерисан број
- atoi — цео број конвертован из стринга
- atof — реалан број конвертован из стринга
- ...

Ključne reči u jeziku C++ su sledeće:

and	and_eq	asm	auto
bitand	bitor	bool	break
case	catch	char	class
compl	const	const_cast	continue
default	delete	do	double
dynamic_cast	else	enum	explicit
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	not	not_eq	operator
or	or_eq	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	xor	xor_eq

Delovi C++ programa

- C++ program se sastoji od objekata, funkcija, promenljivih, konstanti i drugih delova. Ako posmatramo, računajući na to da već imamo dovoljno iskustva u pisanju programa na jeziku C, jednostavan program koji ima zadatak da na ekranu ispiše poruku " Zdravo svete ":

- ***lin 1.*** ***#include <iostream>***
- ***lin 2.*** ***using namespace std;***
- ***lin 3.*** ***int main()***
- ***lin 4.*** ***{***
- ***lin 5.*** ***cout <<" Zdravo svete\n";***
- ***lin 6.*** ***return 0;***
- ***lin 7.*** ***}***

Delovi C++ programa

- U liniji 1. datoteka iostream je uključena u program. Prvi karakter u liniji je # znak što predstavlja signal predprocesoru. Svaki put kada pokrenemo prevodilac predrocesor se izvršava. Predprocesor analizira izvorni kod, tražeći linije koje počinju sa # i deluje na te linije pre nego što se prevodilac pokrene. Ovo je GLAVA PROGRAMA.
- *Include* je instrukcija predrocesoru koja kaže, "Ono što sledi je naziv datoteke. Nađi tu datoteku i postavi je na ovo mesto." Znaci manje i veće oko naziva datoteke ukazuju predprocesoru da potraži tu datoteku na svim uobičajenim mestima. Ukoliko je vaš prevodilac pravilno konfigurisan, znaci < i > će ukazati predrocesoru da potraži datoteku iostream u direktorijumu koji sadrži sve h datoteke za vaš prevodilac. Datoteka iostream (Input-Output-Stream) se koristi od strane naredbe cout, koja pomaže pri ispisivanju na ekran. Rezultat linije 1. je da uključi datoteku iostream u program, kao da ste je sami ukucali. Fajl iostream je samo jedan deo header fajlova koji sadrži rutine standardne biblioteke.
- *Predprocesor* (eng, *preprocessor)* se izvršava pre prevodioca, svaki put kada se prevodilac pozove. Predprocesor prevodi svaku liniju koja počinje znakom # u specijalnu komandu, pripremajući našu izvornu datoteku za prevodilac.

Delovi C++ programa

- U liniji 2. Pozivaju se sve standardne biblioteke potrebne za rad programa.
- Linija 3. označava stvarni početak programa funkcijom main( ). Svaki C++ program ima main( ) funkciju. Generalno, funkcija je blok koda, koja obavlja jednu, ili više akcija. Obično funkcije pozivaju druge funkcije, ali funkcija main( ) je posebna. Kada se naš program pokrene, main( ) se poziva automatski. Poput ostalih funkcija, main ( ) mora da iskaže koju vrednost će vratiti. Tip povratne vrednosti za main( ) u ZDRAVO.CPP je void, što znači da ova funkcija neće vratiti nikakvu vrednost, ali može biti int ili float.
- Sve funkcije počinju otvorenom velikom zagradom ({) i završavaju se zatvorenom velikom zagradom (}). Zagrade za main( ) funkciju su u linijama 4 i 7. Sve između otvorene i zatvorene velike zagrade se smatra kao deo funkcije - TELO PROGRAMA.
- Glavni deo programa je u liniji 5. Objekat cout se koristi za štampanje poruke na ekran. Evo kako se cout koristi: otkucajte reč cout, iza koje treba da se nađe operator izlazne redirekcije (<<). Sve što se nalazi iza operatora izlazne redirekcije biće odštampano na ekranu. Ukoliko želite da odštampate niz karaktera, postavite ga izmedu dvostrukih navodnika ("... "), kao što je prikazano u liniji 5.

Delovi C++ programa

```
//Kratak pogled na cout

#include <iostream>
using namespace std;
int main( )
{
cout << "Zdravo!\n";
cout << "Ovo je 5: " << 5 << "\n";
cout << "Manipulator endl nas prevodi u novi red" << endl;
cout << "Evo veoma velikog broja: \t" << 10000 << endl;
cout << "Ovo je zbir 8 i 5: \t\t" << 8+5 << end1;
cout << "Evo razlomka: \t\t\t" << (float) 5./8. << endl;
cout << "I veoma velikog broja: \t\t" << (double) 7000*7000 << endl ;
cout << "Student Ime i prezime je C++ programer! \n";
return 0;
}
```

Delovi C++ programa

- **ANALIZA:**
- U liniji 3. naredba #include<iostream> dovodi do uključivanja datoteke iostream u izvorni kod. Ovo je potrebno ako konstite cout i srodne funkcije. Sve ostale datoteke pozivaju se sa ***using namespace std;***
- U liniji 6. je jednostavan primer upotrebe cout za štampu niza karaktera. Znak /n je specijalni znak za formatiranje. On ukazuje cout da štampa znak za novi red na ekran. Tri vrednosti su prosleđene za cout u liniji 7 i svaka vrednost je odvojena operatorom izlazne redirekcije <<. Prva vrednost je string: "Ovo je 5: ". Obratite pažnju na prazno mesto posle dve tačke. Prazno mesto je deo stringa. Sledeća vrednost je 5, koja se prosleđuje operatoru izlazne redirekcije, kao i znak za *novi red* (eng. new line character) /n (uvek pod dvostrukim ili jednostrukim navodnicima). Sve to dovodi do štampanja linije na ekran Ovo je 5: 5 . Pošto nema znaka za novi red posle prvog stringa sledeća vrednost se odmah zatim štampa. Ovo se naziva *konkatenacija* (eng. *concatenating)* dve vrednosti.

Delovi C++ programa

- U liniji 8. štampa se poruka sa informacijom na ekran i zatim se posle nje koristi manipulator endl. Namena endl je štampanje znaka za prelazak u novi red na ekranu.
- U liniji 9. uvodimo /t novi znak za formatiranje. On umeće *tab karakter* i koristi se u linijama 8. do l2. da bi se poravnao prikaz na ekranu. Linija 9. pokazuje da mogu biti štampane ne samo celobrojne vrednosti, *već* i *dugačke celobrojne vrednosti* (eng. long integer).
- Linija 10. je primer jednostavnog sabiranja unutar cout. Vrednost od 8+5 je prosleđena cout. ali je 13 odštampano.
- U liniji 11. za cout je prosleđeno *5./8.* Termin (float) je pokazao cout da želite da se ovo izračuna kao decimalni broj, pa je zato odštampan razlomak.
- U liniji 12. za cout je prosleđeno 7000*7000 a termin (double) se koristi da ukaže da želite prikaz rezultata u eksponencijalnom zapisu.
- U liniji 13. napisali ste uz svoje ime i prezime da je C ++ programer.

Delovi C++ programa

- **Na sledećem primeru vidimo upotrebu naredbi za ulaz - izlaz i komentare:**

```
// Program Zad2.cpp
#include <iostream>
using namespace std;
int main()
{
// Ucitavanje prve promenljive
cout << "a= ";
double a;
cin >> a;
// Ucitavanje druge promenljive
cout << "b= ";
double b;
cin >> b;
// Postavljanje parametra p
double p;
p = 2*(a+b) ;
// Postavljanje parametra s
double s;
s = a*b;
// Stampanje p i s
cout << "p= " << p << endl;
cout << "s= " << s << endl;
return 0;
}
```

Delovi C++ programa

- Sledi primer programa koji zahteva unos dva cela broja sa tastature, a koji zatim ispisuje njihov zbir:

```
#include <iostream>
using namespace std;
int main()
{
int broj_1, broj_2;
cout << "Unesite prvi broj: ";
cin >> broj_1;
cout << "Unesite drugi broj: ";
cin >> broj_2;
cout << "Zbir brojeva" << broj_1 << " i " << broj_2
<< " glasi " << broj_1 + broj_2 << endl;
return 0;
}
```

Komentari

- **Kada pišete program, sve je uvek jasno i, samo po sebi, razumljivo je šta želite da uradite. Zanimljivo je, medutim, da kada se mesec dana kasnije vratite programu, isti kod može biti krajnje zbunjujući i nejasan. Nisam siguran kako se to uvlači u program, ali uvek tako biva.**

- ***Tipovi komentara***
- **C++ komentari se mogu pojavljivati u dva oblika:**

  kao dupla kosa crta (//… u jednom redu) i

  kao kosa crta -zvezdica (/*.....*/ u više redova).

  Komentar u obliku duple kose crte, koji ćemo pominjati kao komentar u C++ stilu, ukazuje prevodiocu da ignoriše sve što sledi iza komentara, sve do kraja linije.
- **Komentar kosa crta -zvezdica ukazuje prevodiocu da ignoriše sve što ga sledi, sve dok ne naiđe na oznaku kraja komentara zvezdica -kosa crta (*/). Ovaj komentar ćemo pominjati kao komentar u C stilu. Svaki /* mora imati zatvarajući */.**
- **Kako ste i sami možda predpostavili, komentari u C stilu se koriste i u programskom jeziku C, dok komentari u C++ stilu nisu deo zvanične definicije C-a. Mnogi C++ programeri koriste stalno komentare u C++ stilu u svojim programima, dok komentare u C stilu koriste samo za izbacivanje velikih blokova koda u programu. Možete uključiti komentare u C++ stilu unutar bloka koda, stavljenog u komentar u C stilu: sve, uključujući i komentare u C++ stilu, ignoriše se unutar oznaka komentara u C stilu.**

Korišćenje komentara

- Kao generalno pravilo, ceo program treba da ima komentare na početku, koji vam govore šta program radi. Svaka funkcija, takođe, treba da ima komentar za objašnjenje šta funkcija radi i koje vrednosti vraća. Na kraju, svaki deo Vašeg programa koji je složen treba komentarisati.
- Sledeći listing demonstrira korišćenje komentara pokazujući da oni ne utiču na izvršavanje programa, niti na njegov izlaz:

- ***1: //Program za objašnjenje korišćenja komentara***
- ***2: #include<iostream>***
- ***using namespace std;***
- ***3: int main( )***
- ***4: {***
- ***5: /* Ovo je komentar***
- ***6: i nastavlja se sve do zatvarajuće***
- ***7: oznake komentara zvezdica-kosa crta */***
- ***8: cout<< "Zdravo svete!!!!!" <<endl;***
- ***9: // Ovaj komentar se zavrsava krajem reda***
- ***10: cout << "Ovaj komentar je zavrsen!!!! "<<endl;***
- ***11:***
- ***12: // Dupla kosa crta, kao komentar, može biti sama u liniji***
- ***13: /* kao i kosa crta-zvezdica */***
- ***14: return 0;***
- ***15: }***

Komentari na početku svake datoteke

- Dobra je ideja pisati blok komentara na početku svake datoteke sa kodom, koju napišete. Stvarni stil tog bloka komentara je stvar ličnog ukusa. ali svako takvo zaglavlje treba da uključi bar sledeće informacije:

- naziv funkcije ili programa
- naziv datoteke
- šta ta funkcija ili program rade
- opis kako program radi
- ime autora
- istorijat revizija (beleška o svakoj promeni)
- koji prevodioci, povezivači i drugi alati su konšćeni pri izradi progama)
- dodatne beleške po potrebi.
-
- Veoma je važno obezbediti ažurnost beleški i opise ažurnim. Standardni problem sa zaglavljima je što se zanemare posle početnog kreiranja i tokom vremena postanu više smetnja nego pomoć. Ali, ako se propisno ažuriraju, ona mogu biti od neprocenjive pomoći u razumevanju celog programa.

Obratite pažnju na tačku i zarez!

Verovatno najčešćšća greška početnika u jezicima C i C++ jeste izostavljanje znaka tačka i zarez (;), ali se dešava da se nađe i tamo gde je izlišan. Pravilo za korišćšćenje tačke i zareza je sledeće: svaki iskaz završite tačkom i zarezom osim u slučaju:

- pretprocesorske komande, kao što su #include i #define,
- Složenog iskaza: U praksi, ovo znači da tačku i zarez ne pišete posle završne vitičaste zagrade } izuzev ako je to kraj klase ili deklaracije promenljive.
- Jednostavan program prikazan u produžetku ilustruje ovo pravilo i oba izuzetka.

```
#include <stdio.h>

void main () {
      int  i = 5, j, k = 1;

      while (i > 0) {
            k = k * i;
            i = i - 1;
      }

      printf("k is %d", k);
}
```

Ovo se ne završava tačkom i zarezom jer je pretprocesorska komanda.

Otvorena i zatvorena vitičasta zagrada se ne završavaju tačkom i zarezom (osim ako je u pitanju kraj deklaracije klase).

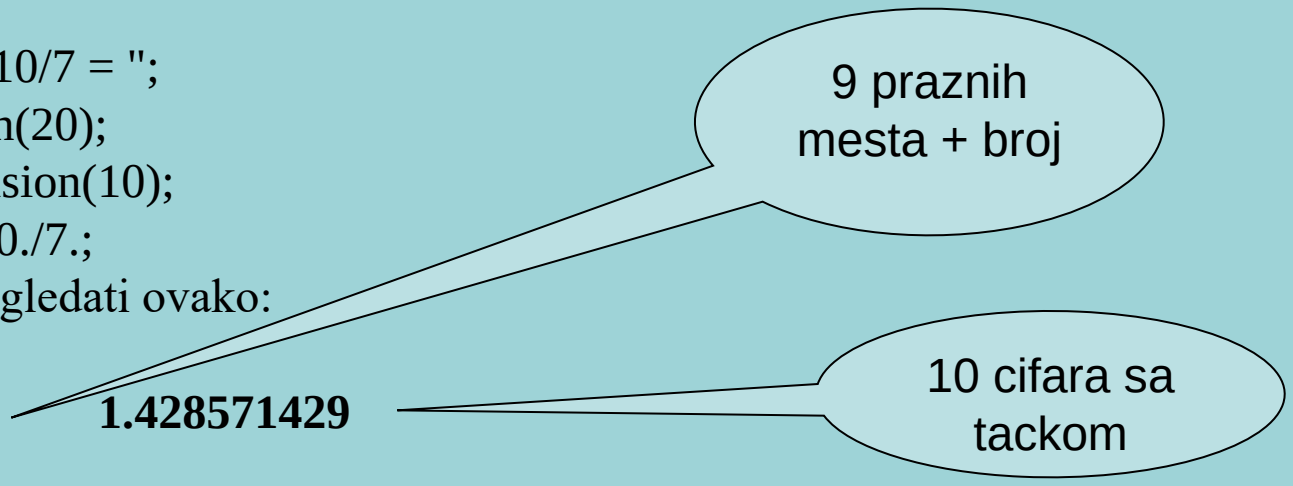
Formatiranje ispisa

- Poznato je da funkcija “printf” nasleđena iz jezika C poseduje obilje mogućnosti za formatiranje ispisa. Slične mogućnosti mogu se ostvariti pomoću objekata izlaznog toka, samo na drugi način. Funkcija **“width”** primenjena nad objektom “cout” ***postavlja željenu širinu ispisa podataka*** (širina se zadaje kao argument), dok funkcija **“precision”** postavlja ***željenu preciznost ispisa (tj. broj tačnih cifara)*** prilikom ispisa realnih podataka. Tako, na primer, ukoliko izvršimo sledeću sekvencu naredbi:

```
cout << "10/7 = ";
cout.width(20);
cout.precision(10);
cout << 10./7.;
```

ispis će izgledati ovako:

10/7 = 1.428571429



- Ovde treba obratiti pažnju da “20” nije *broj razmaka* koji se ispisuju ispred podatka, nego *broj mesta koje će zauzeti podatak*. Kako u našem slučaju podatak zauzima ukupno 11 mesta (10 cifara i decimalna tačka), on se dopunjuje sa 9 dodatnih razmaka ispred, tako da će ukupna širina ispisa biti 20 mesta. Za slučaj kada je zadana širina ispisa manja od minimalne neophodne širine potrebne da se ispiše rezultat, funkcija ”width” se ignoriše.

Formatiranje ispisa

```
cout << "Prihod: "
cout.width(5);
cout << prihod << endl ;
cout << "Oporezovani prihod: ";
cout.width(5);
cout << oporezovani_prihod << endl;
cout << "Poreska dažbina: ";
cout.width(5);
cout << porez << endl;
cout << "Čisti prihod: ";
cout.width(5);
cout << prihod - porez << endl;
```

Izlaz:

Prihod: 11000
Oporezovani prihod: 2000
Poreska dažbina: 666
Čisti prihod: 10334

Formatiranje ispisa

- Biblioteka “iomanip” (skraćeno od engl. *input-output Manipulators*) poseduje tzv. *manipulatore* (odnosno *manipulatorske objekte*) poput “setw”.

- Manipulatori su specijalni objekti koji se *šalju* na izlazni tok (pomoću operatora “<<”) sa ciljem podešavanja osobina izlaznog toka. Njihovom upotrebom, širinu ispisa možemo postavljati “u hodu”, bez prekidanja toka, pomoću naredbi poput:

```
cout << "Prihod:                " << setw(5) << prihod << endl;
cout << "Oporezovani prihod:     " << setw(5) << oporezovani_prihod << endl;
cout << "Poreska dažbina:        " << setw(5) << porez << endl;
cout << "Čisti prihod:           " << setw(5) << prihod_porez << endl;
```

- Naravno, za tu svrhu treba pomoću direktive “#include” uključiti zaglavlje biblioteke “iomanip” u program.

Formatiranje ispisa

\b	**povratnik**
\f	**form feed**
\n	**nova linija**
\r	**carriage return**
\t	**horizontalni tab**
\v	**vertikalni tab**
**\ **	**obrnuta kosa crta**
\"	**dvostruke navodnice**
\ '	**jednostruke navodnice**
\ (carriage return)	**nastavak linije**
\ nnn	**vrednost karaktera**

Tabela Escape karaktera

Tipovi podataka

- **Tipovi podataka**

- **Tipovi podataka određuju moguće vrednosti koje podaci mogu da imaju i moguće operacije koje mogu da se izvode nad tim podacima. Podaci mogu da budu *prosti* (skalarni, nestrukturirani) ili *složeni* (strukturirani). Prosti podaci ne mogu da se dele na manje delove koji bi mogli nezavisno da se obraduju, složeni podaci sastoje se od nekoliko elemenata koji i sami mogu da budu prosti ili složeni.**

- **Od prostih tipova jezik C++ poznaje samo numeričke tipove: Medu njima razlikuju se celi brojevi i realni brojevi.**

- **Osnovne celobrojne tipove čine tipovi *char* i *int*. Varijante tih tipova označavaju se dodavanjem modifikatora ispred ovih oznaka:**
- ***unsigned char, signed char, short int, long int, unsigned int, unsigned short int i unsigned long int.***
- **Ako se koristi modifikator, reč int može da se izostavi (na primer: *short* znači *short int*).**

Tipovi podataka

Ime/ Simbol formata	Dužina u bajtima	Opis	Dijapazon
char/ %c	1	Znak ili broj dužine 8 bita	signed:-128 -127 unsigned: 0-255
short/ %d	2	Celi broj od 16 bita	-32763 do 32762 0 do 65535
long/ %d	4	Celi broj od 32 bita	-2147483648 do 2147483647 0 do 4294967295
int/ %d	16,32,64 ? System	Celi broj	vidi short i long
float/ %f	4	Broj sa dec. zarezom	3.4 e±38 (7 cifre)
double/ %f	8	Broj sa dec. zarezom dvostruke tačnosti	1.7e±308(15 cifre)
long double/ %f	10	Broj sa dec. zarezom višestruke tačnosti	1.2e±4932(19 cifri)
bool	1	logički	true ili false

Tipovi podataka

- Osnovne realne tipove čine tipovi *float* (jednostruka tačnost) i *double* (dvostruka tačnost), a jedina varijanta tih tipova označava se sa *long double* (višestruka tačnost).

  U programskom jeziku C++ ***promenljiva*** (engl. variable) je mesto za čuvanje informacija određeno svojim imenom i lokacijom u memoriji računara. Ime promenljive, npr. mojaPromenljiva može zauzeti zavisno od veličine (tipa) jednu ili više memorijskih adresa.

- Kada definišete promenljivu u programskom jeziku C ++ , morate da prevodiocu date do znanja o kojem tipu promenljive je reč: celobrojnoj (integer), znakovnoj (character) ili nekoj drugoj. Ove informacije saopštavaju prevodiocu koliko prostora da rezerviše i koju vrstu vednosti želite da skladištite u svojoj promenljivoj.

Tipovi podataka

- *//Primer za prezentaciju velicine odredjenih podataka u bajtima preko funkcije* ***sizeof()***
- *1:#include <iostream.h>*
- *2:*
- *3: int main()*
- *4:{*
- *5: cout << "Velicina tipa int je: \t\t" << sizeof(int) <<bajta. \n";*
- *6: cout << "Velicina tipa short je: \t\t" << sizeof(short) <<bajta. \n";*
- *7: cout << "Velicina tipa long je: \t\t" << sizeof(long) <<bajta. \n";*
- *8: cout << "Velicina tipa char je: \t\t" << sizeof(char) <<bajta. \n";*
- *9: cout << "Velicina tipa float je: \t\t" << sizeof(float) <<bajta. \n";*

  10: cout << "Velicina tipa double je: \t\t" << sizeof(double) <<bajta. \n";

  11:

  12: return 0;

  13: }
-
- **IZLAZ:**
- **Velicina tipa int je: 2 bajta.**
- **Velicina tipa short je: 1 bajt.**
- **Velicina tipa long je: 4 bajta.**
- **Velicina tipa char je: 1 bajt.**
- **Velicina tipa float je: 4 bajta.**
- **Velicina tipa double je: 8 bajta.**

KONSTANTE

- U jeziku C, konstantne vrednosti se obično definišu pomoću preprocesorske direktive “#define”:

#define BrojStudenata 50
#define PI 3.141592654

- Ovaj način posjeduje mnoge nedostatke. Najveći nedostatak je što preprocesorski elementi ne podležu sintaksnoj kontroli i tipizaciji jezika, tako da eventualna greška u njihovoj definiciji često ostaje neotkrivena, ili bude otkrivena tek prilikom njihove upotrebe. Stoga, u jeziku C++ ovaj način treba izbegavati. Generalno, preprocesor “nije u duhu” C++-a, tako da njegovu upotrebu u C++-u treba smanjiti na najnužniji minimum (po mogućnosti, samo na direktivu “#include”). Umesto toga, u C++-u konstante deklarišemo kao i promjenljive, uz dodatak ključne reči “**const**”:

const int BrojStudenata = 50;
const double PI = 3.141592654;

- S obzirom da se kod konstanti može govoriti samo o *inicijalizaciji* (a ne o *dodeli*), uputnije je koristiti sintaksu koja ne koristi znak dodele, nego zagrade:

const int BrojStudenata(50);
const double PI(3.141592654);

- Konstante moraju biti inicijalizirane, tako da deklaracija poput **const int** broj; nije dozvoljena.

Logičke promenljive

- Recimo nekoliko stvari o tipu “**bool**”. U jeziku C vrednosti “tačno” i “1” odnosno vrednosti “netačno” i “0” su poistovećene). Dugo vremena (sve do pojave ISO 98 C++ standarda) ista konvencija je važila i u jeziku C++. Međutim standard ISO 98 C++ uveo je dve nove ključne reči “**true**” i “**false**” koje respektivno predstavljaju vrednosti “tačno” odnosno “netačno”. Tako je vrednost svakog tačnog izraza “**true**”, a vrednost svakog netačnog izraza “**false**”. Uvedena je i ključna riječ “**bool**” kojom se mogu deklarisati promjenljive koje mogu imati samo vrednosti “**true**” odnosno “**false**”. Na primer, ako imamo sledeće deklaracije:

bool u_dugovima, punoletan, polozio_ispit;

tada su sasvim ispravne sledeće dodele (uz pretpostavku da takođe imamo deklarisane brojčane promjenljive “stanje_kase” i “starost”):

u_dugovima = stanje_kase < 0;
punoljetan = starost >= 18;
polozio_ispit = **true**;

Pobrojani tipovi - enum

- Na ovom mestu dobro je reći nešto o *pobrojanim* (engl. *enumerated*) *tipovima*. Pobrojani tipovi su postojali i u jeziku C, ali su oni predstavljali samo manje ili više prerušene celobrojne tipove. Njihov tretman u jeziku C++ je kompletno izmenjen, pri čemu su, nažalost, morale nastati izvesne nekompatibilnosti sa jezikom C. U jeziku C++ pobrojani tipovi predstavljaju prvi korak ka *korisnički definisanim tipovima*. Pobrojani tipovi opisuju *konačan skup vrednosti koje su imenovane i uređene* (*tj. stavljene u poredak*) *od strane programera*. Definišemo ih pomoću deklaracije "**enum**", iza koje sledi *ime tipa koji definišemo* i *popis mogućih vrednosti tog tipa unutar vitičastih zagrada* (kao moguće vrednosti mogu se koristiti proizvoljni *identifikatori* koji nisu već iskorišteni za neku drugu svrhu). Na primer, pomoću deklaracija

- **enum** Dani {Ponedeljak, Utorak, Sreda, Cetvrtak, Petak, Subota, Nedelja};

  identično kao celobrojne konstante iz sledeće deklaracije:
- **const int** Ponedeljak(0), Utorak(1), Sreda(2), Cetvrtak(3),Petak(4), Subota(5), Nedelja(6);

  enum Rezultat {Poraz, Nerešeno, Pobeda};

  definišemo dva nova *tipa* nazvana "Dani" i "Rezultat". Promenljive pobrojanog tipa možemo deklarisati na uobičajeni način, na primer:
- Rezultat danasnji_rezultat;
- Dani danas, sutra;

Kompleksni brojevi

- Kompleksni brojevi predstavljaju korisno obogaćenje jezika C++, koje je ušlo u standard ISO 98 jezika C++. Kompleksni tip spada u tzv. *izvedene tipove* (engl. *derived types*) i predstavljen je rečju "complex". Kako ovaj tip nije ugrađen u samo jezgro C++ nego je definisan u standardnoj biblioteci jezika C++, za njegovo korišćenje potrebno je uključiti u program zaglavlje biblioteke koje se takođe zove "complex". U matematici je kompleksni broj formalno definisan kao par realnih brojeva, ali kako u jeziku C++ imamo nekoliko tipova za opis realnih brojeva ("**float**", "**double**" i "**long double**"), kompleksne brojeve je moguće izvesti iz svakog od ovih tipova. Sintaksa za deklaraciju kompleksnih promjenljivih je

  complex<*osnovni_tip*> *popis_promjenljivih*;

  gde je *osnovni_tip* tip iz kojeg izvodimo kompleksni tip. Na primer, deklaracijom oblika

  complex<**double**> z; deklarišemo kompleksnu promjenljivu "z" čiji su realni i imaginarni deo tipa "**double**". Kompleksni tipovi se mogu izvesti čak i iz nekog od celobrojnih tipova. Na primer, complex<**int**> gauss; deklariše kompleksnu promjenljivu "gauss" čiji realni i imaginarni brojevi mogu biti samo celobrojne vrednosti tipa "**int**". Inače, kompleksni brojevi čiji su realni i imaginarni dio celi brojevi imaju veliki značaj u teoriji brojeva i algebri, gdje se nazivaju *Gaussovi celi brojevi* (to je i bila motivacija da ovu promjenljivu nazovemo "gauss").

  complex<**double**> z1(2, 3.5);

Sledeći primer prikazuje program za rešavanje kvadratne jednačine, uz upotrebu kompleksnih promenljivih:

```
#include <iostream.h>
#include <math.h>
#include <complex.h>
int main()
{
double a, b, c;
cout << "Unesi koeficijente:\n";
cin >> a >> b >> c;
double d = b * b – 4 * a * c;
if(d >= 0) {
double x1 = (-b - sqrt(d)) / (2 * a);
double x2 = (-b + sqrt(d)) / (2 * a);
cout << "x1 = " << x1 << "\nx2 = " << x2 << endl;
}
else {
complex<double> x1 = (-b – sqrt(complex<double>(d))) / (2 * a);
complex<double> x2 = (-b + sqrt(complex<double>(d))) / (2 * a);
cout << "x1 = " << x1 << "\nx2 = " << x2 << endl;
}
return 0;
}
```

VEKTORI

- Možemo reći da rad sa C-nizovima, mada sasvim legalan, nije “u duhu” jezika C++. Da bi se ovi problemi izbegli, u standard ISO 98 jezika C++ uveden je novi tip podataka, nazvan “vector”, koji je definisan u istoimenom zaglavlju standardne biblioteke jezika C++ (tako da za korištenje ovog tipa podataka moramo uključiti u program zaglavlje biblioteke “vector”). Ovaj tip podataka (zovimo ga prosto *vektor*) zadržava većinu svojstava koji poseduju standardni nizovi, ali ispravlja neke njihove nedostatke. Promenljive tipa “vector” mogu se deklarisati na nekoliko načina, od kojih su najčešći sledeći:

  vector< *tip_elemenata* > *ime_promenljive*;
  vector< *tip_elemenata* > *ime_promenljive*(*broj_elemenata*);
  vector< *tip_elemenata* > *ime_promenljive*(*broj_elemenata*, *inicijalna_vrednost*);

- Na primer, vektor “ocene”, čiji su elementi celobrojni, možemo deklarisati na jedan od sledećih
- načina:

```
vector<int> ocene;
vector<int> ocene(10);
vector<int> ocene(10, 5);
```

```
student[20]
PetBroja[5]={10,20,30,40,50}
```

- Prva deklaracija deklariše vektor “ocene”, koji je inicijalno prazan, odnosno ne sadrži niti jedan element (videćemo kasnije kako možemo naknadno dodavati elemente u njega). Druga deklaracija (koja se najčešće koristi) deklariše vektor “ocene”, koji inicijalno sadrži 10 elemenata, a koji su automatski inicijalizirani na vrednost 0. Treća deklaracija deklariše vektor “ocene”, koji inicijalno sadrži 10 elemenata, a koji su automatski inicijalizirani na vrednost 5.

VIŠEDIMENZIONALNI VEKTORI

- Vektori, o kojima smo govorili na prethodnom predavanju, lepo se mogu iskoristiti za kreiranje *dvodimenzionalnih struktura podataka*, poput *matrica*, i još opšitije, *višedimenzionalnih struktura podataka*. Tako se, na primer, matrice mogu formirati kao *vektori čiji su elementi vektori*. Tako, recimo, matricu sa *m* redova i *n* kolona možemo formirati kao vektor od *m* elemenata čiji su elementi vektori od *n* elemenata. To je najlakše uraditi ovako:

- **vector<vector<*tip_elemenata*>>*ime_matrice*(*m*,vector<*tip_elemenata*>(*n*));**
- Na primer, matricu "a" sa 10 redova i 5 kolona, čiji su elementi realni brojevi tipa "**double**", možemo deklarisati ovako:

- **vector<vector<double> > Polje(5, vector<double>(3)); ili**
- **Int Polje[5][3]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}**
- **Int Polje[5][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12},{13,14,15}}**

- Ova prividno rogobatna sintaksa zahtijžževa pažljiviju analizu. Prvo, mi ovim zapravo deklarišemo *vektor čiji su elementi tipa* "vector<**double**>", odnosno vektor vektora realnih brojeva, a to nam upravo i treba. Primetimo razmak između dva znaka ">" u ovoj deklaraciji. Prvi parametar, "10", kao što znamo, predstavlja dimenziju vektora, drugi parametar predstavlja *vrednost koja se koristi za inicijalizaciju elemenata vektora*, konačno dobijamo *vektor od 5 elemenata, pri čemu je svaki element vektor od 3 elementa*, što nije ništa drugo nego tražena matrica formata 5 × 3.

VIŠEDIMENZIONALNI VEKTORI

```
//kreiranje visedimenzionalnog niza:
#include<iostream.h>
int main()
{int SomeArray[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};
for(int i = 0; i < 5; i++)
for(int j = 0; j < 2; j++)
{
cout << “SomeArray[" << i  << “][" << j << “]: ";
cout << “SomeArray[i][j]<<endl;
}
return 0;
}
```

Izlaz:

```
SomeArray[0][0]:0
SomeArray[0][1]:0
SomeArray[1][0]:1
SomeArray[1][1]:2
.
.
SomeArray[4][0]:4
SomeArray[4][1]:8
```

Program unosi sa tastature matricu sa zadanim brojem redova i kolona i vraca je kao rezultat

```
vector<vector<double> > UnesiMatricu(int br_redova, int br_kolona)
{
vector<vector<double> > m = KreirajMatricu(br_redova, br_kolona);
for(int i = 0; i < br_redova; i++)
for(int j = 0; j < br_kolona; j++)
{
cout << "Element (" << i + 1 << "," << j + 1 << "): ";
cin >> m[i][j];
}
return m;
}
// Ispisuje zadanu matricu
void IspisiMatricu(vector<vector<double> > m)
{
for(int i = 0; i < BrojRedova(m); i++)
for(int j = 0; j < BrojKolona(m); j++)
cout.width(10);
cout << m[i][j];
}cout << endl;}
}
```

KARAKTERI I STRINGOVI

- Sada ćemo preći na tretman stringova u jeziku C++. Kao što od ranije znamo, u C-u se stringovi predstavljaju kao obični nizovi čiji su elementi *znakovi* (*karakteri*), tj. tipa "**char**", pri čemu se specijalni znak sa ASCII šifrom 0 koristi kao oznaka kraja stringa (zbog toga obično govorimo o *nul-terminiranim nizovima znakova*). Sve manipulacije sa stringovima posmatranim kao nul-terminiranim nizovima znakova rade i dalje bez ikakvih izmena u C++-u. Međutim, u C++-u je uveden novi tip podataka, nazvan "string", koji nudi znatne olakšice u radu sa stringovima u odnosu na pristup nasleđen iz C-a. Stoga se u C++-u preporučuje korišćenje tipa "string" kad god se zahteva elegantna i fleksibilna manipulacija sa stringovima.

KARAKTERI I STRINGOVI

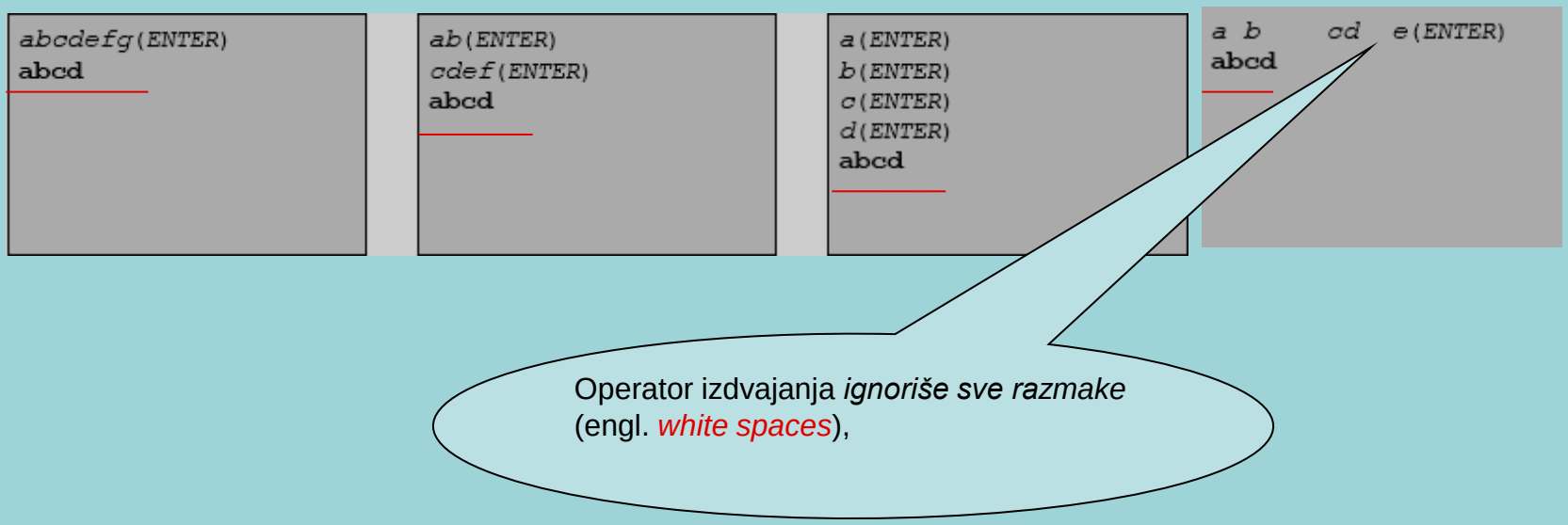
- Promjenljive tipa "**char**" se automatski ispisuju kao znakovi, bez potrebe da to naglašavamo i na kakav način. Sledeća sekvenca naredbi ilustruje ove činjenice:

  char prvi, drugi, treci, cetvrti;

  cin >> prvi >> drugi >> treci >> cetvrti;

  cout << prvi << drugi << treci << cetvrti << endl;

- Sledeća slika prikazuje neke od mogućih scenaria izvršavanja ove sekvence:



KARAKTERI I STRINGOVI

//Program za demonstraciju rada sa karakternim podacima

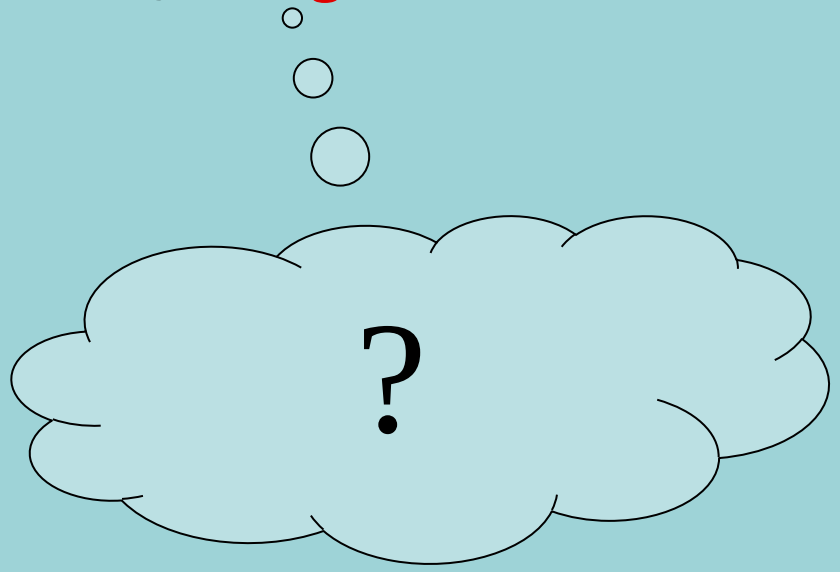
```
#include<iostream.h>
int main()
{
char recenica[80];
cout << "Unesite string: ";
cin>>recenica;
cout << “Prikaz unetog sadrzaja: " << recenica << endl;
return 0;
}
```

Ulaz:
Unesite string: Zdravo svete
Izlaz:
Prikaz unetog sadrzaja: Zdravo

KARAKTERI I STRINGOVI

Objašnjenje programa za demonstraciju rada sa karakternim podacima:

Bafer je definisan kao niz od 80 karaktera. On prihvata 79 karaktera dugačak string i null karakter za terminiranje. U liniji 6 korisnik unosi string koji se prenosi u bafer. Ako korisnik unese string duži od 79 karaktera višak se neće videti. Kada korisnik unese prazno mesto cin će to shvatiti kao kraj stringa i prestaće da upisuje u bafer. Rešenje ovog problema je funkcija cin.get.



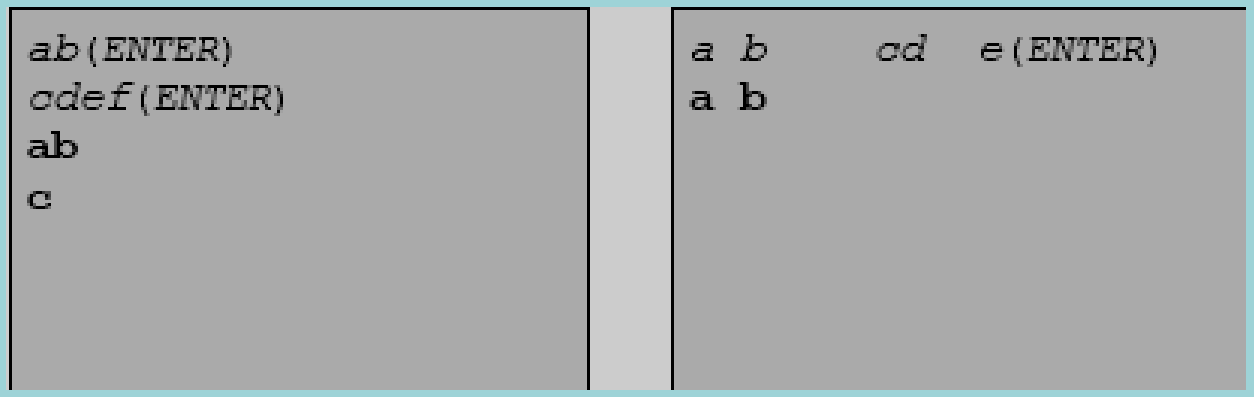
KARAKTERI I STRINGOVI

- Ukoliko iz bilo kojeg razloga želimo da izdvojimo znak iz ulaznog toka, bez ikakvog ignorisanja, *ma kakav on bio* (uključujući razmake, specijalne znake tipa oznake novog reda '\n', itd.) možemo koristiti poziv funkcije "get" bez parametara nad objektom ulaznog toka. Ova funkcija izdvaja sledeći znak iz ulaznog toka, ma kakav on bio, i vraća ga kao rezultat (ako je ulazni tok prazan, ova funkcija zahteva da se ulazni tok napuni svežim podacima). Da bismo shvatili kako deluje funkcija "cin.get()", razmotrimo sledeću sekvencu naredbi:

```
char prvi, drugi, treci, cetvrti;
prvi = cin.get();
drugi = cin.get();
treci = cin.get();
cetvrti = cin.get();
cout << prvi << drugi << treci << cetvrti << endl;
```

KARAKTERI I STRINGOVI

- Dva moguća scenaria izvršavanja ovog programa prikazana su na sledećoj slici:



- U prvom scenariju, nakon izvršenog unosa, promenljive “prvi” i “drugi” sadrže znakove “a” i “b”, promenljiva “treci” sadrži *oznaku za kraj reda*, dok promenljiva “cetvrti” sadrži znak “c”, koji sledi iza oznake kraja reda. Sada je sasvim jasno zbog čega ispis ovih promenljivih daje prikazani ispis. U drugom scenariju, nakon izvršenog unosa, promenljive “prvi” i “drugi”, “treci” i “cetvrti” sadrže respektivno znak “a”, razmak, znak “b” i ponovo razmak.

KARAKTERI I STRINGOVI

//Međutim, ako proširimo naredbu sa parametrima promenljive, njene dužine i po potrebi definišemo terminator, onda je:

```
#include<iostream.h>
int main()
{
char recenica[80];
cout << "Unesite string: ";
cin.get(recenica,79); //uzima do 79 karaktera ili novog reda - return
cout << “Prikaz unetog sadrzaja: " << recenica << endl;
return 0;
}
```

Ulaz:

Unesite string:Zdravo svete

Izlaz:

Prikaz unetog sadrzaja: Zdravo svete

KARAKTERI I STRINGOVI

- Bilo kakva iole složenija manipulacija sa tekstualnim podacima bila bi izuzetno mukotrpna ukoliko bi se oni morali obrađivati isključivo *znak po znak*. Stoga je pomoću funkcije **“getline“** nad objektom ulaznog toka moguće učitati čitav niz znakova *odjedanput*. Ukoliko se koristi sa *dva parametra*, tada prvi parametar predstavlja *niz znakova u koji se smještaju pročitani znakovi*, dok drugi parametar predstavlja broj koji je *za jedinicu veći od maksimalnog broja znakova koji će se pročitati*.

```
char recenica[50];
cin.getline(recenica, 51);
```

- Ukoliko se funkcija “getline“ pozove sa *tri parametra*, tada prva dva parametra imaju isto značenje kao i u prethodnom slučaju, dok treći parametar predstavlja *znak koji će se koristiti kao oznaka završetka unosa*. Drugim rečima, čitaće se svi znakovi sve dok se ne pročita navedeni znak ili dok se ne pročita onoliko znakova koliko je zadano drugim parametrom. Tako će poziv funkcije

```
cin.getline(recenica, 51, '.');
```

zatražiti unos sa ulaznog uređaja, a zatim pročitati najviše 49 znakova iz ulaznog toka u niz “recenica”, pri čemu se čitanje prekida ukoliko se pročita znak '.' (tačka).

KARAKTERI I STRINGOVI

- Na primer, sledeća sekvenca koristi dobro poznatu funkciju “strlen” iz biblioteke “cstring” za određivanje dužine niza karaktera:

```
char a[100];
cout << "Unesi neki niz znakova (ne više od 99 znakova): ";
cin.getline(a, 100);
cout << "Uneli ste " << strlen(a) << " znakova.";
```

Ili:

```
char recenica[100];
cout << "Unesi neku rečenicu: ";
cin.getline(recenica, 100);
cout << "Unesena rečenica glasi: " << recenica;
```

Whitespace eater- “gutač praznina”

- “Gutač praznina” (engl. *c* ili *whitespace extractor*), je u jeziku C++ imenovan prosto imenom “ws”.
- Drugim rečima, nakon izvršenja izraza poput “cin >> ws” eventualne praznine sa početka ulaznog toka biće uklonjene.

```
Unesi broj: 123(ENTER)
Unesi tekst: ABC(ENTER)

Broj: 123 Tekst: ABC
```

```
int broj;
char tekst[100];
cout << "Unesi broj: ";
cin >> broj >> ws;
cout << "Unesi tekst: ";
cin.getline(tekst, 100);
cout << "\nBroj: " << broj << "Tekst: " << tekst << endl;
```

Promjenljive tipa “string”

- Promenljive tipa “string”, koje za razliku od nul-terminiranih nizova znakova možemo zvati *dinamički stringovi*, deklarišu se na uobičajeni način, kao i sve druge promenljive (pri tome reč “string”, poput “complex” ili “vector”, nije ključna reč). Za razliku od običnih nizova znakova, pri deklaraciji promenljivih tipa “string” ne navodi se maksimalna dužina stringa, s obzirom da se njihova veličina automatski prilagođava tokom rada. Promenljive tipa “string”, ukoliko se eksplicitno ne inicijaliziraju, automatski se inicijaliziraju na *prazan string* (tj. string koji ne sadrži niti jedan znak, odnosno string dužine 0), Dakle, deklaracija

  string s;

  deklariše promenljivu “s” tipa “string” koja je automatski inicijalizirana na prazan string.-

```
string s = "Ja sam string!";
cout << s << endl;
s = "A sada sam neki drugi string...";
cout << s << endl;
```

Funkcije tipa “strcpy() i strncpy()”

- Ovo su nasleđene f-je iz bibliotele C jezika I služe za kopiranje jednog stringa u drugi:
  strcpy(string2,string1) i
  strncpy(string2,string1,MaxLength)

Obavezno zahtevaju pozivanje hederske funkcije string.h.

```
#include<iostream.h>
#include<string.h>
int main()
{
char String1[]=“Covek nije ostrvo”;
char String2[80];
strcpy(String2,String1);
cout << “String1 izgleda: " << String1 << endl;
cout << “String2 izgleda: " << String2 << endl;
return 0;
}
```

Izlaz:

String1 izgleda: Covek nije ostrvo
String2 izgleda: Covek nije ostrvo

Međutim, ako je dužina String2 manja od dužine String1 kopiranje neće biti uspešno! Zato koristimo:

Funkcije tipa “strcpy() i strncpy()”

```
//Funkcija STRNCPY
#include<iostream.h>
#include<string.h>
int main()
{
cons int MaxLength=80
char String1[]=“Covek nije ostrvo”;
char String2[MaxLength+1];
strncpy(String2,String1,MaxLength);
cout << “String1 izgleda: " << String1 << endl;
cout << “String2 izgleda: " << String2 << endl;
return 0;
}
```

Izlaz:

String1 izgleda: Covek nije ostrvo
String2 izgleda: Covek nije ostrvo

Funkcije tipa “strlen()”

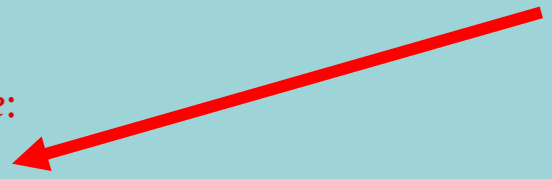
- Najčešće korišćena funkcija koja vraća dužinu nul-terminiranog niza je strlen().

```
//Funkcija STRLEN
#include<iostream.h>
#include<string.h>
int main()
{
char buffer[80];
cout << "Unesite string do 80 karaktera: ";
cin.get(buffer,80); //uzima do 79 karaktera ili novog reda - return
cout << “Vaš string je: “ << strlen(buffer) << “ karaktera dug “ << endl;
return 0;
}
```

ALGEBRA KARAKTERA

- Primjenom operatora “+”konstrukcije bi se mogle napisati na sledeći način (i to bez potrebe za pisanjem posebne funkcije):

- str3 = str1 + str2 + str3;
- str3 = str1 + "bla bla";
- str3 = "bla bla" + str1;
- str3 = znakovi_1 + str1;
- U posljednje tri konstrukcije dolazi do automatskog pretvaranja operanda koji je tipa “niz znakova” u tip “string”. Ipak, za primenu operatora “+” *barem jedan od operanada* mora biti dinamički string.



- Stoga, sedeće konstrukcije *nisu legalne*:
- str3 = "Dobar " + "dan!";
- str3 = znakovi_1 + "bla bla";
- str3 = znakovi_1 + znakovi_2;

- Naravno, kao i u mnogim drugim sličnim situacijama, problem je rešiv uz pomoć *eksplicitne konverzije tipa*, na primer, kao u sledećim konstrukcijama (moguće su i brojne druge varijante):
- str3 = znakovi_1 + string("bla bla");
- str3 = string(znakovi_1) + znakovi_2;

Operatori i izrazi

Tip operacije	Operacija	Operator
Multiplikovane operacije	stepenovanje-korenovanje	pow(x,y)
	množenje	*
	deljenje	/
	ostatak deljenja	%
Aditivne operacije	sabiranje	+
	oduzimanje	-
Unarne operacije	plus	+
	minus	-
Relacije	manje	<
	manje ili jednako	<=
	veće	>
	veće ili jednako	>=
	jednako	=
	nejednako	đ=

Operatori predstavljaju radnje koje se izvršavaju nad operandima dajući pri tome određene rezultate. Izrazi su proizvoljno složeni sastavi operanada i operatora.

Aritmetički operatori

- *Aritmetički operatori* služe za izvođenje osnovnih aritmetičkih operacija. U njih spadaju binarni operatori za sabiranje (+), oduzimanje (-), množenje (*), deljenje (/) i nalaženje ostatka deljenja celih brojeva (%).
- Unarni operator + nema nikakvog efekta, a služi za izmenu algebarskog znaka broja. U jeziku C++ postoje još dva unarna operatora(inkrementa i dekrementa) za povećavanje (++) i smanjivanje (--) vrednosti operanda za jedan. Ova dva operatora mogu da budu napisana ispred operanda (prefiksna notacija) kada je vrednost izraza nova vrednost operanda, ili iza operanda (postfiksna notacija) kada je vrednost izraza vrednost operanda pre promene.

Relacijski operatori

- *Relacijski operatori služe za upoređivanje numeričkih podataka i daju logičke vrednosti.* U jeziku C++ to znači celobrojnu vrednost 0 za logičku neistinu, odnosno celobrojnu vrednost 1 za logičku istinu. Ti operatori se obeležavaju sa
- **== (jednako),**
- **! = (različito),**
- **< (manje),**
- **<= (manje ili jednako),**
- **> (veće) i**
- **>= (veće ili jednako).**

Logički operatori

- *Logički operatori služe za izvođenje logičkih operacija nad logičkim podacima dajući logički rezultat.* U jeziku C++ nulta vrednost operanda smatra se logičkom neistinom, a bilo koja nenulta vrednost logičkom istinom. Rezultati su 0 za logičku neistinu i isključivo 1 za logičku istinu.
- Postoje unarni operator za logičku *ne(!)* operaciju i binarni operatori za logičku *i* (&&) i *ili* (||) operaciju. U slučaju operatora && prvo se izračunava vrednost prvog operanda i ako je to =0, rezultat je 0 i vrednost drugog operanda uopšte se ne izračunava. u slučaju operatora || prvo izračunava se vrednost prvog operanda i ako je to različit od 0, rezultat je 1 i vrednost drugog operanda se uopšte ne izračunava.

Operatori po bitovima

- *Operatori po bitovima* od standardnih viših programskih jezika postoje samo u jeziku C i jezika koji su proizašli iz jezika C *(C++, Java)*. Oni služe za manipulacije sa bitovima unutar celobrojnih podataka.
- Postoje unarni operator za komplementiranje bit po bit (~) i binarni operatori za logičke operacije *i* (&), *uključivo ili* (|) i *isključivo ili* (^), bit po bit. Pored toga postoje binarni operatori za pomeranje binarne vrednosti prvog operanda ulevo (<<) i udesno (>>) za broj mesta koji je jednak vrednosti drugog operanda.

Uslovni operator

- *Uslovni operator* (?:) je specifičan ternarni operator jezika C. lma tri operanda i piše se u obliku *a?b:c*. Prvo se izračunava vrednost izraza a i ako ima vrednost logičke istine (različito od 0), izračunava se vrednost izraza *b* i to predstavlja vrednost celog izraza. Ako a ima vrednost logičke neistine (=0), izračunava se vrednost izraza c i to predstavlja rezultat celog izraza. Bitno je da se od izraza *b* i c uvek izračunava samo jedan.

Adresni operatori

- *Adresni operatori* služe za manipulisanje sa adresama podataka. U užem smislu, tu spadaju unarni operatori za nalaženje adrese datog podatka (&) i za posredni pristup podatku pomoću pokazivača (*) - koji se naziva i operatorom indirektnog adresiranja. U širem smislu u adresne operatore spadaju i aditivni aritmetički operatori. Naime, dozvoljeno je na vrednost nekog pokazivača (adrese) dodati celobrojnu vrednost, odnosno od nje oduzeti celobrojnu vrednost. Jedinica mere promene adrese u tim slučajevima je veličina pokazivanih podataka. Drugim rečima ako pokazivač p pokazuje na neku komponentu datog niza, tada je p+l pokazivač na narednu, a p-l na prethodnu komponemu tog niza. Na kraju, dozvoljeno je i upoređivati vrednosti dva pokazivača relacijskim operatorima, s tim da upotreba operatora <, <=, > i >= ima smisla samo ako oba operanda pokazuju na komponente istog niza.

Operatori inkrementiranja i dekrementiranja

- **Operatori inkrementiranja i dekrementiranja:**
- Operator inkrementiranja ++ i dekrementiranja -- su unarni operatori istog prioriteta kao i unarni. Oba operatora primenjuju se isključivo na promenljive i javljaju se u prefiksnom i postfiksnom obliku.

- Operator inkrementiranja: + + promenljivoj p dodaje vrednost 1 , pa važi:
- **p++ je ekvivalentno p=p+1**
- **++ p je ekvlvalentno p=p+1**

- Operator dekrementiranja -- promenljivoj p oduzima vrednost 1, pa važi:
- **p-- je ekvivalentno p = p-1**
- **--p je ekvivalenlno p = p-1**

Ako u nekom izrazu postoji p++ (*postfiksni oblik*) vrednost promenljive se prvo koristi u izrazu, pa tek onda inkrementira. U slučaju da u izrazu postoji ++p (prefiksni oblik) promenljiva se prvo inkrementira, pa se tek onda njena vrednost koristi u izrazu.

- Slično prethodnom, ako u izrazu postoji p-- (*postfiksni oblik*) vrednost se prvo koristi u izrazu, pa tek onda dekrementira, a u slučaju --p (prefiksni oblik) vrednost se dekrementira i tako ažurirana koristi u izrazu.

Operatori inkrementiranja i dekrementiranja

```
/*Program za prikaz inkrementa i dekremanta */
#include<iostream.h>
main()
{
int a = 0, b = 0, c = 0;
cout<<"a=“<<a<< " b=" <<b<< "c=“<< c<<endl;
 a = ++b + ++c;
cout<<"a=“<<a<< " b=" <<b<< "c=“<< c<<endl;
a = b++ + c++;
cout<<"a=“<<a<< " b=" <<b<< "c=“<< c<<endl;
a = ++b + c++;
cout<<"a=“<<a<< " b=" <<b<< "c=“<< c<<endl;
a = ++c + c;
cout<<"a=“<<a<< " b=" <<b<< "c=“<< c<<endl;
}
```

Izlaz

```
a=0 b=0 c=0
a=2 b=1 c=1
a=2 b=2 c=2
a=5 b=3 c=3
a=8 b=3 c=4
```

ФУНКЦИЈЕ

- Сличност са математичким функцијама:

✓ потребно је именовати их и предати им податке (аргументе)

✓ реализују операције над подацима

✓ добија се резултат

- Програми комбинују:

✓ функције из стандардних библиотека и

✓ функције дефинисане у самом програму

ФУНКЦИЈЕ ОМОГУЋУЈУ

- Једноставно пројектовање програма

лакше је савладати програм у деловима

- Једноставно тестирање рада програма

лакше је откривање грешака по деловима програма

- Неограничено коришћење у програмима

постојеће функције могу постати делови других програма

- Ефикасно коришћење меморије

нема понављања наредби функције у меморији

- Апстракција

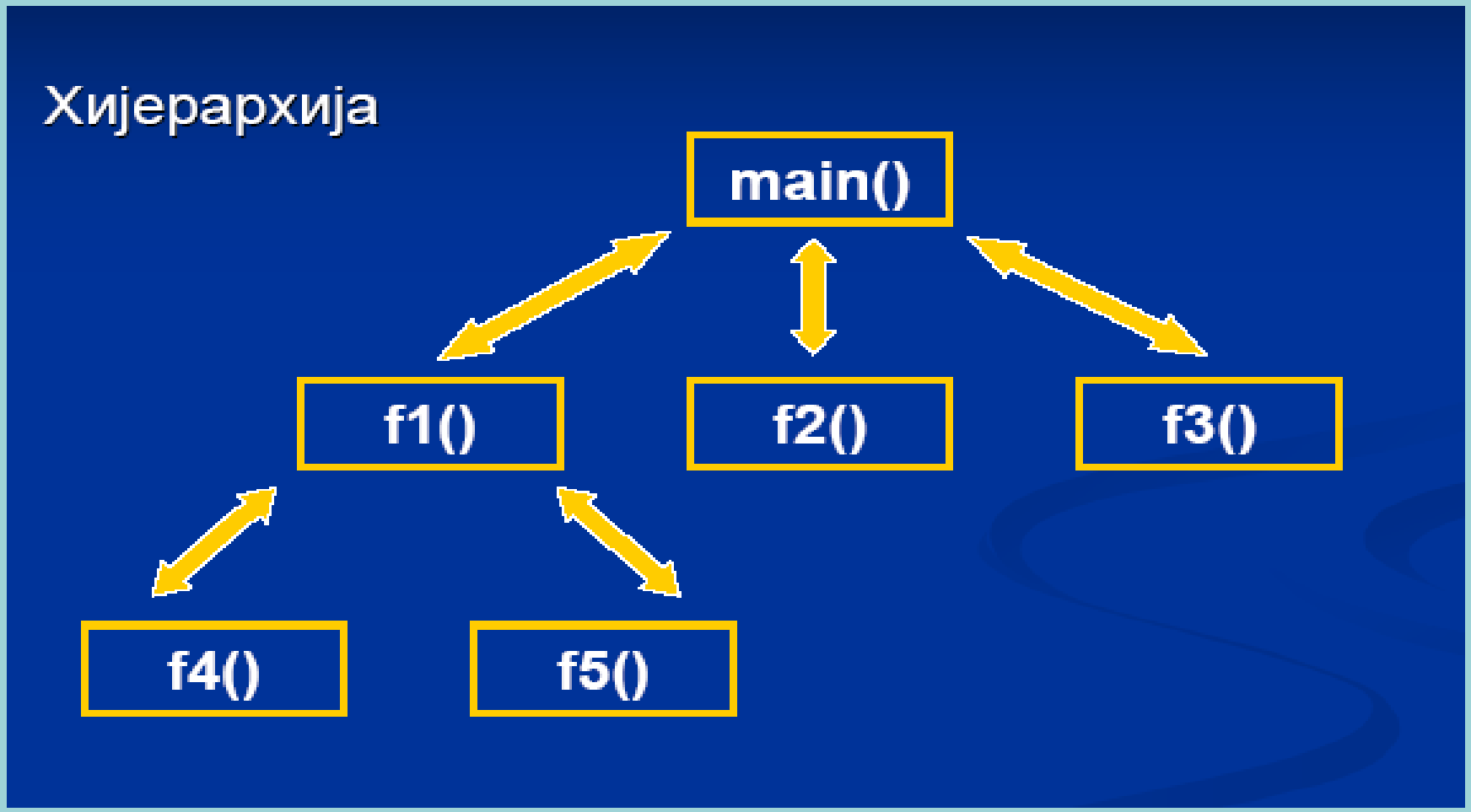
свака функција скрива детаље свог решења

Особине функција

Свака функција, као један део програма:

- Има свој дефинисани задатак
- Користи једне податке, решава свој задатак и даје друге резултујуће податке
- Ако је написана по свим постојећим правилима, скрива своје решење
- Ако нису испоштована правила за писање функције, долази до грешке

Размена података између функција



Подаци у функцијама

- Листа аргумената и повратна вредност

✓ један начин размене података између појединих функција једног програма

- Глобалне променљиве

✓ други начин размене података између појединих функција једног програма

- Локалне променљиве

✓ декларисане унутар функције

✓ скривене унутар функције

Листа аргумената и повратна вредност функције

- Листа аргумената
- Листа података које функција користи са места свог позива у програму (може их бити : 0, 1 или више)
- Повратна вредност
- Вредност податка који функција враћа на место свог позива у програму (једна или ниједна)

Формална листа аргумената функције

- У дефиницији функције:

```
tip_povratne_vrednosti  ime ( lista_argumenata )
{
        naredbe deklaracije
        izvrsne naredbe
        naredba povratka
}
```

за сваки од аргумената:
тип и формално име

Повратна вредност од функције

- У дефиницији функције

```
tip_povratne_vrednosti  ime ( lista_argumenata )
{
        naredbe deklaracije
        izvrsne naredbe
        naredba povratka
}
```

подразумева се: int
ако је нема: void

Повратна вредност од функције

- У дефиницији функције

```
tip_povratne_vrednosti  ime ( lista_argumenata )
{           naredbe deklaracije
            izvrsne naredbe
            return (povratna_vrednost);
}
```

могућа је само једна повратна вредност

Тело (саме наредбе) функције

- У дефиницији функције

```
tip_koji_vraca  ime_funkcije (lista_argumenata)
{     naredbe deklaracije
      izvrsne naredbe
      naredba povratka
      }
```

Унутар тела једне функције не може бити дефиниција неке друге функције

Позив функције

- У позиву функције:

ime (lista_argumenata)

за сваки од аргумената:
стварна вредност

Позив функције

- Два начина за позив функције:
- *Позив функције по вредности*

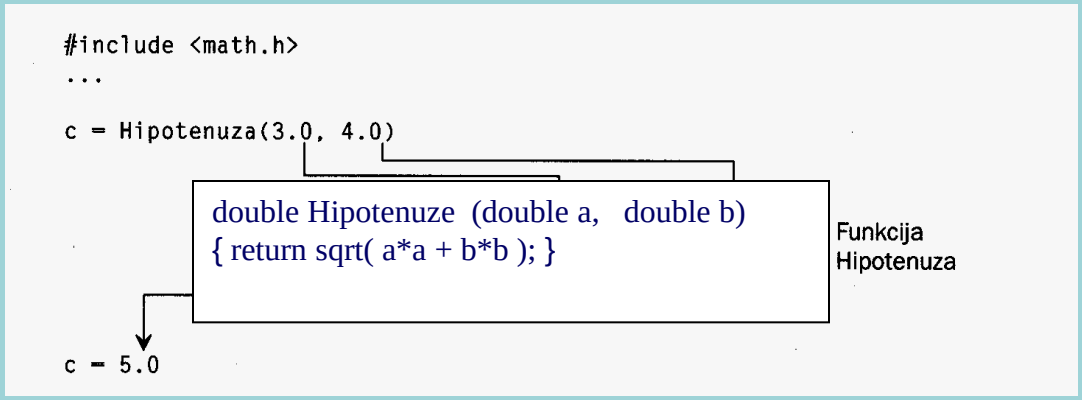
✓ аргумент је вредност основног типа

- *Позив функције по референци*

✓ аргумент је вредност показивача (адреса)

Funkcije

- U C++-u postoji samo jedna vrsta podprogramiranja - funkcija. Funkcije koje ne vraćaju nikakvu vrednost deklarišu se sa *void*, a one koje je vraćaju imaju povratni tip kao što je *int, double* ili *float*. Ovo pojednostavljuje sintaksu C++-a -ne postoje zasebne reči Sub i Function kao u Visual Basicu.
- Sigurno ste se ranije sretali s funkcijama. Funkcija može imati jedan argument ili više njih, a ne mora imati nijedan (u zavisnosti od toga kako je deklarisana) i izračunavanjem daje vrednost koja se potom može koristiti u većem izrazu. Na narednoj slici je prikazano kako bi mogao da radi poziv funkcije Hipotenuza.



- Rezultat izraza Hipotenuza(3.0. 4.0) je poziv funkcije Hipotenuza i prosleđivanje vrednosti dva parametra 3.0 i 4.0. Funkcija koristi iskaz return da bi vratila kontrolu onome ko je pozvao funkciju i da bi vratila rezultat, dužinu hipotenuze -vrednost 5.0.

Opšta sintaksa za funkcije

- Opšta sintaksa u C++ programu kada su u pitanju funkcije sledi po šablonu prikazanom na sledećoj slici.
- Pre nego što funkciju pozovete morate je deklarisati; tome služe *prototipi_funkcije*. Prototip funkcije pruža prevodiocu informacije o tipu tako da on zna koji tip argumenata da očekuje.

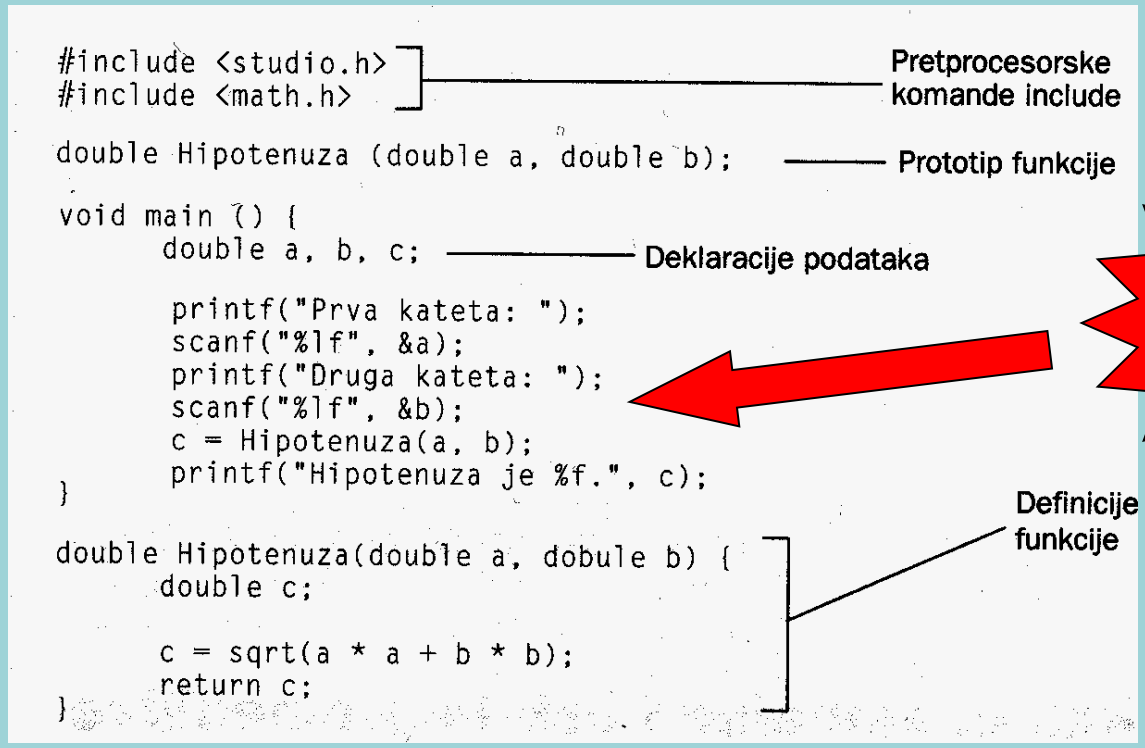
- ***direktive_include i ostale predprocesorske direktive***
- ***prototipi_funkcija ;***
- ***void main()***
- ***{ deklaracije_i_iskazi }***

- ***povratni_tip ime funkcije (argumenti)***
- ***{ deklaracije i iskazi }***

- Format prototipa funkcije izgleda skoro identično kao format prvog reda (zaglavlja) definicije funkcije:
- **povratni_tip ime_funkcije(argumenti) ;**
- *Prototip* funkcije se završava tačkom i zarezom (;).
- Nemojte stavljati tačku i zarez iza završne vitičaste zagrade } *definicije* funkcije. Po tome ih i jeste lako razlikovati.

Opšta sintaksa za funkcije

- Primer: Sintaksa funkcije ima više smisla kada se pokaže na primeru. Slika u nastavku ilustruje svaki deo sintakse funkcije uključujući prototip, poziv funkcije i definiciju funkcije. Prototip priprema za poziv funkcije (tako što javlja prevodiocu za koje tipove treba da proverava), poziv funkcije izvršava funkciju, a definicija funkcije govori programu kako da izvrši funkciju.
- U ovom primeru morate naznačiti dve datoteke zaglavlja (stdio.h i math.h) jer program koristi ulazno-izlazne funkcije (printf i scanf) kao i matematičku funkciju (sqrt), koja vraća kvadratni koren broja. Datoteke zaglavlja imaju prototipe za ove funkcije.



Demonstracija funkcije

```
// Demonstiranje pozivanja funkcije
#include <iostream.h>
void DemonstrationFunction() ;
//Funkcija: Demonstracija funkcije
//štampa korisnu poruku
int main()
{
cout << "Unutar main\n";
DemonstrationFunction();
cout << "Nazad u main\n";
return 0;
}
void DemonstrationFunction()
{cout<< "Mi smo unutar Demonstracione funkcije\n";}
//Funkcija main -štampa poruku, zatim
//poziva DemonstrationFunction, a zatim, štampa
//druqu poruku.
```

Izlaz iz programa je:

Unutar main
Mi smo unutar Demonstracione funkcije
Nazad u main

Korišćenje funkcija

- **Funkcije *vraćaju ili vrednost ili void*, što znači da ne vraćaju ništa. *Funkcija koja sabira dva cela broja može da vrati zbir* i zato bi se definisala da vraća celobrojnu (eng. integer) vrednost. *Funkcija koja samo štampa poruku nema šta da vrati* i zato se deklariše da vraća void.**
- **Funkcija se sastoji od zaglavlja i tela. Zaglavlje se sastoji, po redosledu pojavljivanja, od tipa vrednosti koja se vraća, naziva funkcije i parametara funkcije. Parametri funkcije omogućavaju da se vrednosti proslede funkciji. Zato, ako funkcija treba da sabere dva broja, brojevi bi trebalo da budu parametri funkcije. Tipično zaglavlje funkcije izgleda ovako:**

  int main(int a, int b)
- ***Parametar* funkcije (eng. *parameter)* je deklaracija tipa vrednosti, koja će biti prosleđena funkciji: stvarna vrednost prosleđena pozivanjem funkcije naziva se argument funkcije. Mnogi programeri koriste ova dva pojma, parametre i argumente, kao sinonime. Drugi, s druge strane, vode računa o tehničkoj razlici. U ovoj knjizi termini će biti korišćeni kao sinonimi.**
- **Znači, telo funkcije se sastoji od otvorene velike zagrade, bez ijedne ili sa više naredbi i zatvorene velike zagrade. Naredbe obavljaju posao za koji je funkcija namenjena. Funkcija može da vrati vrednost, koristeći naredbu return. To će, takode, dovesti do kraja rada funkcije. Ukoliko ne stavite naredbu return u svoju funkciiu, ona će, automatski, vratiti void na kraju funkcije. Vrednost koja se vraća mora biti tipa koji je deklarisan u zaglavlju funkcije.**

Korišćenje funkcija

U listingu je prikazana funkcija sa dva celobrojna parametra, koja vraća celobrojnu vrednost. Nemojte brinuti o sintaksi ili specifičnosti rada sa celobrojnim vrednostima (na primer int x).

```
#include <iostream.h>
int Add (int x, int y) ;
int main()
{cout << "Ja sam u main()!\n";
int a, b, c;
cout << "Unesite dva broja: ";
cin >> a;
cin >> b;
cout << "\nPozivam Add()\n";
c=Add(a,b);
cout << "\nNazad u main().\n";
cout << "c ima vrednost " << c;
cout << "\nIzlazim....\n\n";
return 0;}
int Add (int x, int y)
{cout << "U Add(), preuzimam " << x << " i " << y << "\n";
return (x+y);}
```

Izlaz:

```
Ja  sam u main()!
Unesite dva broja: 3 5
Pozivam Add()
U Add(), preuzimam 3 i 5
Nazad u main() .
c ima vrednost 8
Izlazim....
```

Podrazumevani argumenti funkcija

- Prilikom poziva funkcije potrebno je navesti listu argumenata koja odgovara listi argumenata
- u zaglavlju funkcije. C++ dopušta da se neki od argumenata u pozivu funkcije izostave, tako
- da se umesto njih koriste podrazumevane – difolt vrednosti.

- Primer:
- ***#include <iostream.h>***
- ***void funkcija(int a, int b=5)***
- ***{cout << "a = " << a << endl;***
- ***cout << "b = " << b << endl << endl;}***
- ***void main()***
- ***{funkcija (3,4);***
- ***funkcija (10);}***

- **Ispisuje se :**
- ***a = 3***
- ***b = 4***
- ***a = 10***
- ***b = 5***
- **U drugom pozivu funkcije izostavljen je drugi argument, umesto koga se koristi**
- **podrazumevani (b=5).**

Prenos po referenci

- ***Prenos po referenci*** *dolazi do punog izražaja kada je potrebno preneti više od jedne vrednosti iz funkcije nazad na mesto njenog poziva*. Pošto funkcija ne može vratiti kao rezultat više vrednosti, kao rezultat se nameće korišćenje prenosa parametara po referenci, pri čemu će funkcija koristeći reference prosto *smestiti* tražene vrednosti u odgovarajuće stvarne parametre koji su joj prosleđeni. Ova tehnika je ilustrovana u sledećem programu u kojem je definisana funkcija "RastaviSekunde", čiji je prvi parametar broj sekundi, a koja kroz drugi, treći i četvrti parametar prenosi na mesto poziva informaciju o broju sati, minuta i sekundi koji odgovaraju zadanom broju sekundi. Ovaj prenos se ostvaruje tako što su drugi i treći formalni parametar ove funkcije ("sati", "minute" i "sekunde") deklarisani kao reference, koje se za vreme izvršavanja funkcije vezuju za odgovarajuće stvarne argumente:

```
#include <iostream.h>
void RastaviSekunde(int br_sek, int &sati, int &minute, int &sekunde);
 int main()
{int sek, h, m, s;
cout << "Unesi broj sekundi: ";
cin >> sek;
RastaviSekunde(sek, h, m, s);
cout << "h = " << h << " m = " << m << " s = " << s << endl;
return 0;}
void RastaviSekunde(int br_sek, int &sati, int &minute, int &sekunde)
{sati = br_sek / 3600;
minute = (br_sek % 3600) / 60;
sekunde = br_sek % 60;}
```

Primeri funkcija

```
/*Primer 1*/
#include<iostream.h>
void ispis(int i1, int i2, int i3); /* F-ja ne vraca nikakvu vrednost*/
void main()
{
int a,b,c;
a=b=c=1;
ispis(a,b,c);
a=b=c=2;
ispis(a,b,c);
}
void ispis(int i1, int i2, int i3)
{
cout<<"Vrednost promenljive param1 je "<<i1<<endl;
cout<<"Vrednost promenljive param2 je "<<i2<<endl;
cout<<"Vrednost promenljive param3 je "<<i3<<endl;
}
IZLAZ:
Vrednost promenljive param1 je 1
Vrednost promenljive param2 je 1
Vrednost promenljive param3 je 1
Vrednost promenljive param1 je 2
Vrednost promenljive param2 je 2
Vrednost promenljive param3 je 2
```

Primeri funkcija

```
/*Primer 2 kada program poziva dve funkcije*/
#include<iostream.h>
#include<math.h>
void Hipotenuza(void);
void Ucitaj(void);
double a,b,c;/* deklaracija globalne promenljive, pre pocetka programa*/
void main()
{
Ucitaj();
Hipotenuza();
cout<<"Hipotenuza je:"<<c<<endl;
}
void Hipotenuza(void)
{
c=sqrt(a*a+b*b);
}
void Ucitaj(void)
{
cout<<"Prva kateta:";
cin>>a;
cout<<"Druga kateta:";
cin>>b;
}
```

IZLAZ:

Prva kateta:3

Druga kateta:4

Hipotenuza je:5

Primeri funkcija

- /* Primer 3 za izracunavanje $y=\sin^2(x)+\cos^2(x)$*/
- #include<iostream.h>
- #include<math.h>
- double Funkcija(double x);
- void main()
- {
- double x,y; /* deklaracija lokalnih promenljivih, posle pocetka programa*/
- cout<<"Unesite vrednost za x:";
- cin>>x;
- y=Funkcija(x);
- cout<<"Vrednost funkcije je:"<<y<<endl;
- }
- double Funkcija(double x)
- {
- double y;
- y=sin(x)*sin(x)+cos(x)*cos(x);
- return y;
- }

- //ZLAZ: Unesite vrednost za x:90 Vrednost funkcije je 1.0000000

Показивачи - аргументи функције

- **Показивачи аргументи** - доносе функцији адресе променљивих и тако омогућавају функцији:

✓ 1. да мења вредности променљивих од адреса које добије,
✓ 2. да упише више својих резултујућих вредности у променљиве од адреса које добије,
✓ 3. да приступи свим елементима низа било ког типа од адресе почетка низа.

Измена вредности променљиве чију адресу функција добија у аргументу

Формални аргумент (показивач) иницијализује се при позиву функције - на стварну адресу променљиве

```
main ()
{   int a=10;
    duplo(&a);                  /*adresa promenljive &a*/
    printf("%d", a);
}
void duplo(int *ptrx)           /*pokazivac ptrx*/
{   *ptrx = *ptrx * 2;
}
```

Више резултујућих вредности преко преко аргумената функције

- ✓ Сваки формални аргумент (показивач) може бити иницијализован на адресу резултујуће променљиве

```
#include<iostream.h>
#include<math.h>
void duplo(int x, int &ptr1, double &ptr2);
main ()
{ int a=10, rez1; double rez2;
duplo(a, rez1, rez2); /*adrese promenljivih*/
cout<<"Za vrednost promenljive:"<<a<<endl;
cout<<"Prva vracena vrednost je:"<<rez1<<endl;
cout<<"Druga vracena vrednost je:"<<rez2<<endl;
return 0;}
void duplo(int x, int &ptr1, double &ptr2) /*pokazivaci*/
{ ptr1 = x * 2;
ptr2 = x / 2.0; }
```

Правила досега променљивих

Оператор резолуције досега (**::**)

- за приступ унутар функције, глобално декларисаној променљивој која има исто име као локална у тој функцији

- за груписање глобално декларисаних променљивих и њихово коришћење на одређен начин у појединим деловима програма

Правила досега променљивих

```
#include <iostream.h>

void f(void);
int x=10;

main()
{   int x=20;
    cout << " Vrednost lokalno deklarisane x: " << x << endl;
    cout << " Vrednost globalno deklarisane x: " << (::x) << endl;
    return 0;                                                    }
```

```
C:\> Vrednost lokalno deklarisane x: 20
C:\> Vrednost globalno deklarisane x: 10
```

Правила досега променљивих

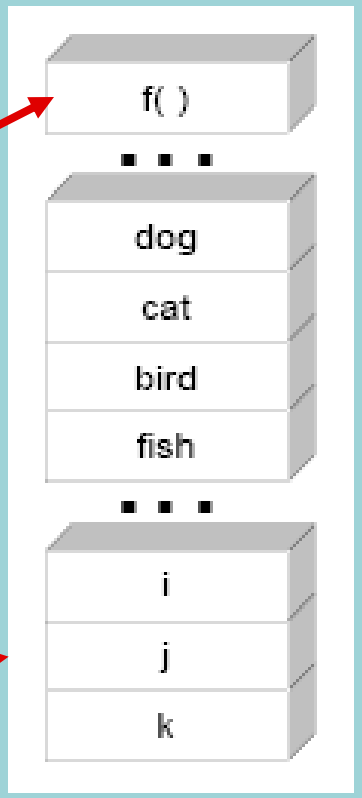
```
/*Vazi isto i u bilo kojoj drugoj funkciji, razlicitoj od glavne*/

void f(void)
{   int x=30;

    cout << " Vrednost lokalno deklarisane x: " << x << endl;
    cout << " Vrednost globalno deklarisane x: " << (::x) << endl;
}
```

```
C:\> Vrednost lokalno deklarisane x: 30
C:\> Vrednost globalno deklarisane x: 10
```

POINTERI-Pokazivači

- Uvek kada se startuje program on se obično sa diska učitava u memoriju računara. Ona je sastavljena od sekvencijalne serije memorijskih lokacija koje imaju svoje adrese a dužine su onoliko bita, kolika je dužina reči *word size* dotične mašine:

```
#include <iostream.h>
void f(int pet)
{cout << "pet id number: " << pet << endl;}
int dog, cat, bird, fish;

int main()
{int i, j, k;}
```



POINTERI-Pokazivači

- Primer:
- int *pa, x;
- int a[20];
- double d;

- Ova naredba koristi adresni operator:

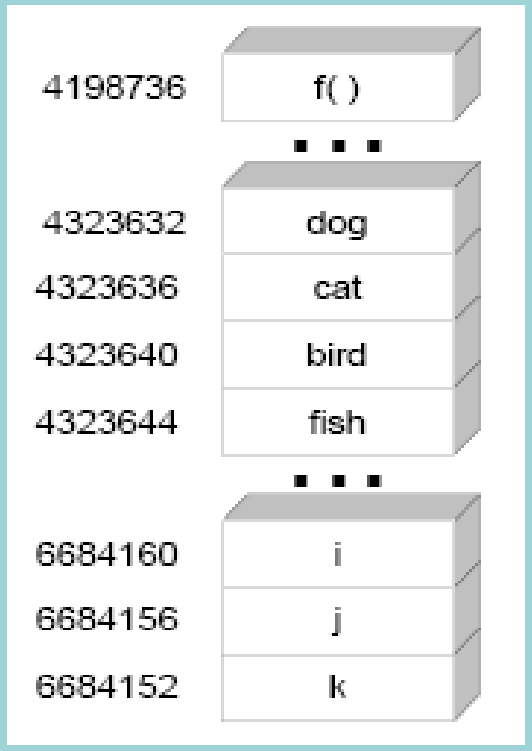
- pa = &a[5];
  Adresni operator (**&**) uzima adresu svih šest elemenata polja a. Ovaj rezultat se smešta u pointer promenljive pa.

  x = *pa;

  Operator indirekcije (*****) se koristi u ovom primeru da pristupi **int** vrednosti na aresi memorisanoj u pa. Ova vrednost se pridružuje intedžer promenljivoj x.

POINTERI-Pokazivači

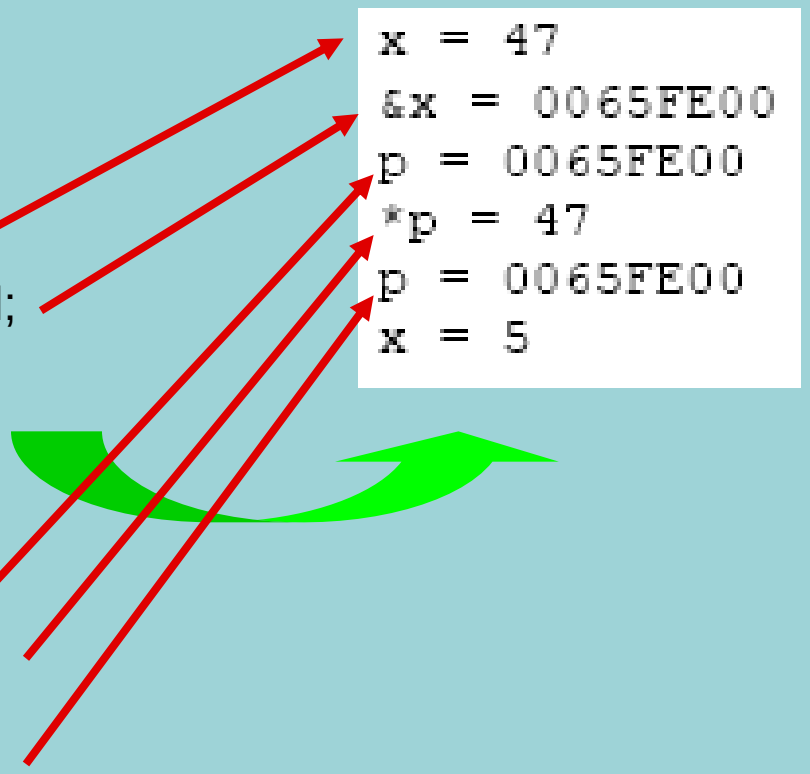
```
#include <iostream.h>
int dog, cat, bird, fish;
void f(int pet)
{cout << "pet id number: " << pet << endl;}

int main()
{
int i, j, k;
cout << "f(): " << (long)&f << endl;
cout << "dog: " << (long)&dog << endl;
cout << "cat: " << (long)&cat << endl;
cout << "bird: " << (long)&bird << endl;
cout << "fish: " << (long)&fish << endl;
cout << "i: " << (long)&i << endl;
cout << "j: " << (long)&j << endl;
cout << "k: " << (long)&k << endl;
}
```



Tip **(long)** je *cast*. On napominje “ne tretiraj ovo kao normalni tip, već ga postavi na tip **long**.”

POINTERI-Pokazivači

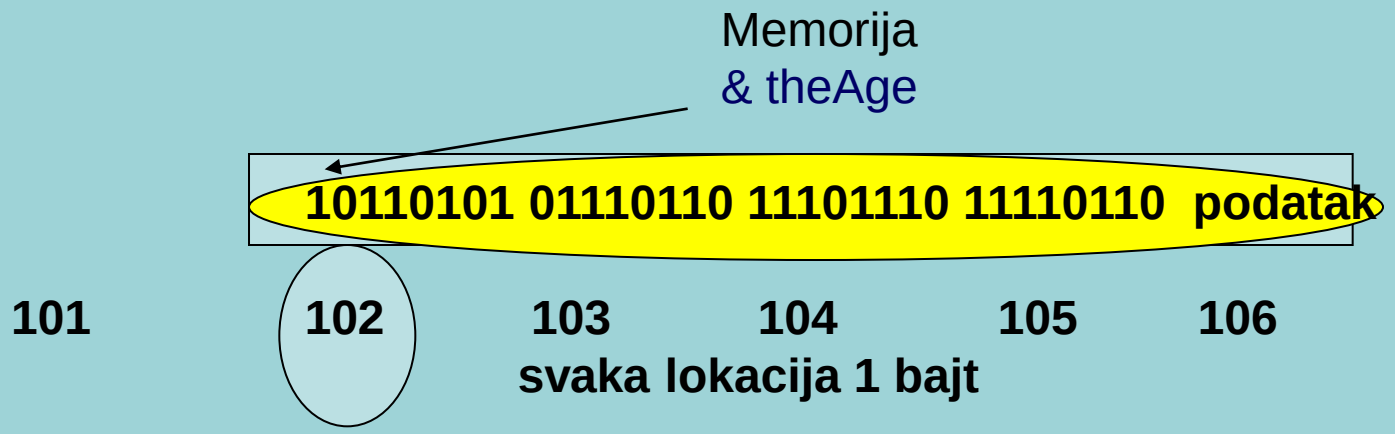
```
#include <iostream.h>
void f(int* p) ;
int main() {
int x = 47;
cout << "x = " << x << endl;
cout << "&x = " << &x << endl;
f(&x);     //poziv funkcije
cout << "x = " << x << endl;
}
void f(int* p) {
cout << "p = " << p << endl;
cout << "*p = " << *p << endl;
*p = 5;
cout << "p = " << p << endl;
}
```



POINTERI-Pokazivači

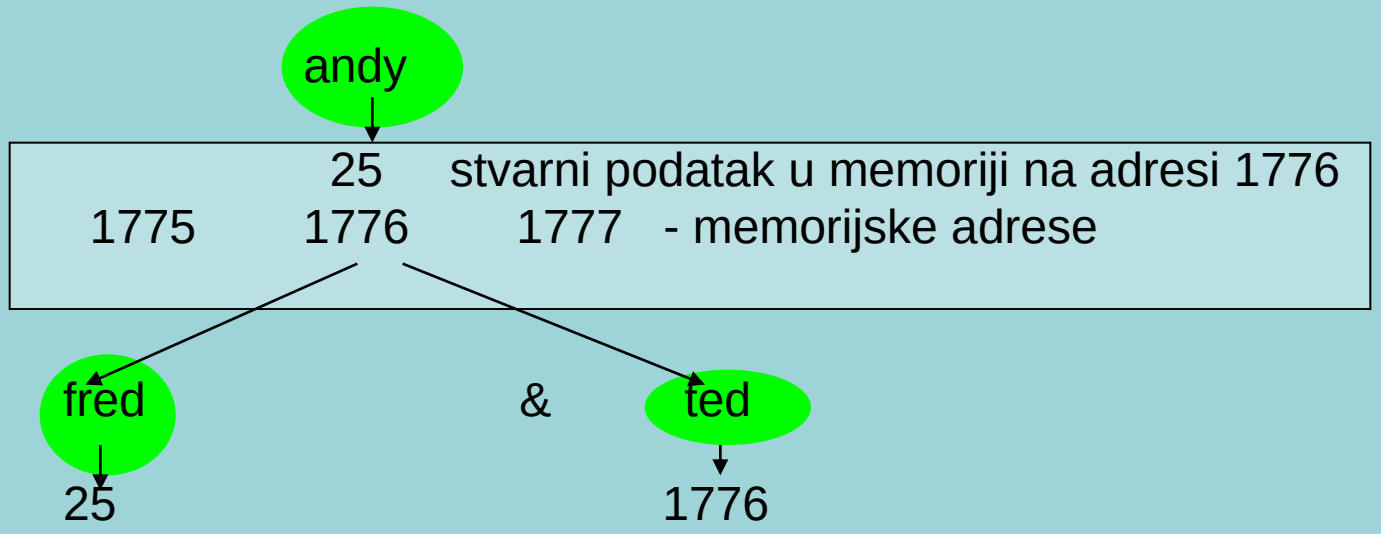
Predstavljaju dakle jedan od najmoćnijih alata koji stoje na raspolaganju C++ programerima, kojima se direktno manipuliše sa memorijom računara. Mogu biti *pokazivači adresa ili pokazivači indirekcije.*

Pokazivač adresa je promenljiva koja čuva memorijsku adresu ćelije ili lvalue. Prikaz promenljive theAge tipa Integer može izgledati:



unsigned long int &theAge = 4 bajta = 32 bita, a ime promenljive &theAge pokazuje na prvi bajt pa je 102 adresa promenljive theAge

POINTERI



- andy=25; //promenljiva andy postavlja se na vrednost 25 i dodeljuje joj se memorijska adresa 1776
- fred=andy; //promenljiva fred uzima vrednost promenljive andy tj. 25
- ted=&andy /*promenljiva ted uzima vrednost adrese promenljive andy a to je 1776*/

- Sintaksa operatora adresa je: ***&<promenljiva>***

POINTERI

```
//Demonstriranje operatora adresa od, i adresa lokalnih promenljivih
#include <iostream.h>
int main()
{
unsigned short shortProm=5;
unsigned long longProm=65535;
long sProm = -65535;
 cout <<"shortVProm:\t" << shortProm;
cout << " Adresa shortProm:\t";
cout << &shortProm<< "\n";
cout << "lonqVProm:\t" << longProm;
cout << " Adresa longProm:\t" ;
cout << &longProm << "\n";
 cout << "sProm:\t"<< sProm;
cout << " Adresa sProm:\t" ;
cout << &sProm << "\n";
return 0;
}
```

Izlaz:

short*Prom*: 5 **Adresa short*Prom*: 0x1b1a**

long*Prom*: 65535 **Adresa long*Prom*: 0x1b16**

s*Prom*r: -65535 **Adresa s*Prom*: 0x1b12**

POINTERI

- Pokazivač *indirekcije sa operatorom* * (naziva se i operator dereferenciranja) omogućuje da i bez znanja specifične adrese date promenljive čuvamo njenu adresu u pokazivaču. Predpostavimo da je how0ld celobrojna promenljiva. Da bi deklarisali pokazivač nazvan pAge za čuvanje njene adrese napisali bi: int *pAge=0. Ovo deklariše promenljivu pAge kao pokazivač na int, što je deklariše i za čuvanje adrese od int. U ovom primeru pAge ima vrednost nula - divlji pokazivači, ali bi uglavnom svi pokazivači morali da imaju vrednost različitu od nule.

- unsigned short int how0ld=50; //kreiranje promenljive
- unsigned short int * pAge=0 ; //kreiranje pokazivača
- pAge=&how0ld; //stavi adresu od how0ld u pAge

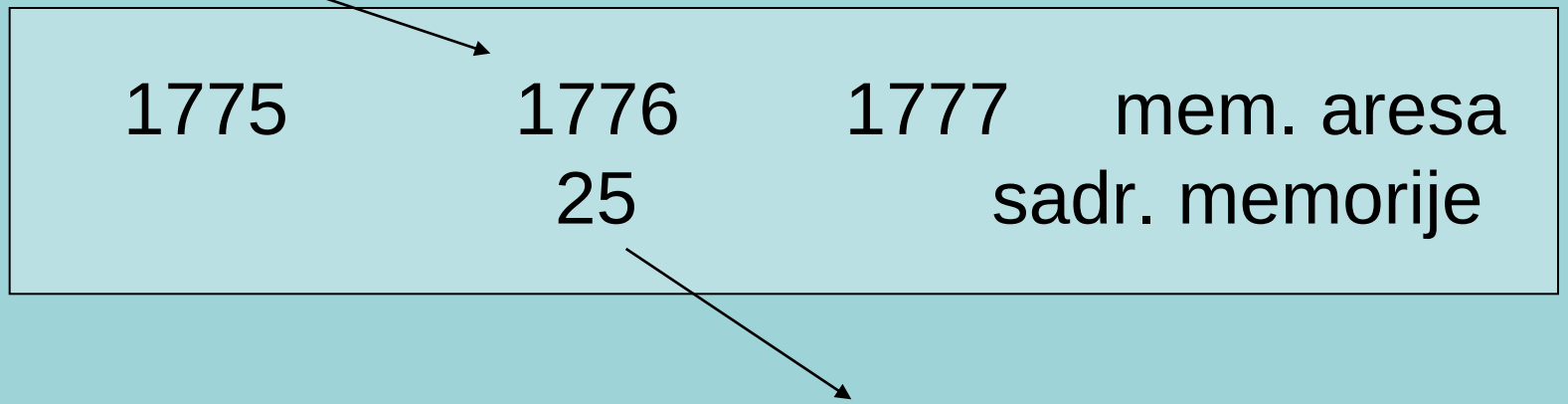
POINTERI

- Prva linija kreira promenljivu how0ld čiji je tip unsigned short int, a inicijalizuje je na vrednost 50.
- Druga linija definiše pAge kao pokazivač za tip unsigned short int i inicijalizuje ga na 0.
- Treća linija dodeljuje adresu promenljive how0ld pokazivaču pAge. Znači korišćenjem pAge možemo odrediti *vrednost promenljive how0ld, pristupajući joj preko pokazivača*. Ovakav pristup se naziva indirekcija. Indirekcija znači pristupanje vrednosti preko adrese, koja se čuva u pokazivaču. On obezbeđuje indirektan način za dobijanje vrednosti, koja se čuva na toj adresi. Ako pogledamo sledeći primer:

- ted=1776; //adresa ted ima vrednost 1776
- beth=ted; //beth je takođe 1776
- beth=* ted; //beth je sada podatak koji se nalazi na adresi 1776 a
- to je vrednost sadržana u njemu, naprimer 25.

POINTERI

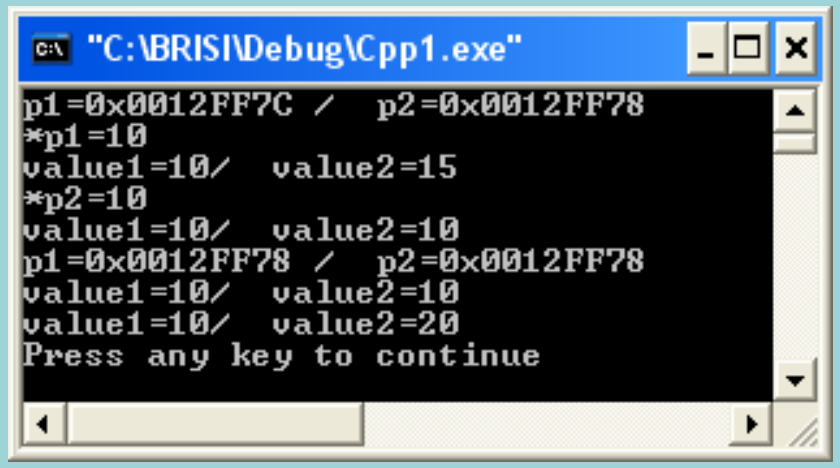
- ted - ima vrednost
- 1776



- beth je podatak na adresi 1776
- a to je 25

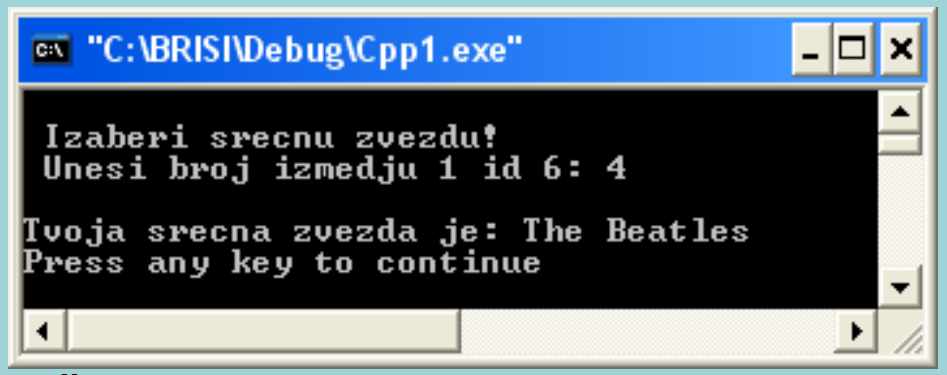
POINTERI

```
#include <iostream.h>
main ()
{
int value1 = 5, value2 = 15;
int *p1, *p2;
p1 = &value1; // p1 = adresa value1
p2 = &value2; // p2 = adresa value2
cout << "p1=" << p1 << " /  p2=" << p2<<"\n" ;
*p1 = 10; // ukazatelj na p1=10
cout<<"*p1="<<*p1<<endl;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
*p2 = *p1; // ukazatelj p2 jednak je ukazatelju p1
cout<<"*p2="<<*p2<<endl;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
p1 = p2; // p1 = p2
cout << "p1=" << p1 << " /  p2=" << p2<<"\n" ;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
*p1 = 20; // ukazatelj na p1=20
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
return 0;
}
```



POINTERI

```
// Inicijalizacija pointera sa stringovima
#include <iostream.h>
int main() { // Inicijalizacija pointerskog polja
char *pstr[ ]= { "Robert Redford",
                 "Sting",
                 "Lassie",
                 "The Beatles",
                 "Boris Karloff",
                 "Oliver Hardy" };
char *pstart = "Tvoja srecna zvezda je: ";
int izbor = 0;
cout << endl << " Izaberi srecnu zvezdu!"<<endl;
cout << " Unesi broj izmedju 1 id 6: ";
cin >> izbor;
cout << endl;
if(izbor >= 1 && izbor <= 6) // Provera ulaznih podataka
cout << pstart << pstr[izbor-1]; // Izabrano ime zvezde
else
// Invalid input
cout << "Zalim, nisi izabrao srecnu zvezdu.";
cout << endl;
return 0; }
```



Stek i slobodno skladište:

- Pomenućemo pet područja memorije:
  - » prostor globalnih imena,
  - » slobodno skladište,
  - » registri,
  - » prostor za kod i
  - » stek.
- Globalne promenljive se nalaze u prostoru globalnih imena. Registri se koriste za interne domaćinske funkcije. Kod se nalazi u prostoru za kod, naravno, kao što je pamćenje vrha steka i pokazivača instrukcija. Lokalne promenljive su na steku, zajedno sa parametrima funkcije. Skoro sva preostala memorija se dodeljuje slobodnom skladištu.
- Problem sa lokalnim promenljivama je taj što one nisu trajne: Po povratku iz funkcije, one se odbacuju. Globalne promenljive rešavaju taj problem po cenu neograničenog pristupa širom programa, što vodi do kreiranja koda koji je težak za razumevanje i održavanje. Stavljanje podataka u slobodno skladište rešava oba ova problema. O slobodnom skladištu možete razmišljati kao o masivnoj sekciji memorije, u kojoj hiljade sekvencijalno označenih kockica leže, čekajući vaše podatke.

Stek i slobodno skladište:

018/55556666

MILAN
Taster 1
A broj ?

Stek i slobodno skladište:

- Stek se automatski čisti po povratku iz funkcije. Sve lokalne promenljive izlaze iz opsega i uklanjaju se sa steka.
- Slobodno skladište se ne čisti sve dok se vaš program ne završi i vaš je zadatak da oslobodite svaki memorijski prostor koji ste rezervisali. Prednost slobodnog skladišta je što memorija koju rezervišete ostaje raspoloživa, dok je eksplicitno ne oslobodite. Ako rezervišete memorju na slobodnom skladištu u funkciji, memorija je još uvek raspoloživa po povratku iz funkcije. Prednost pristupanja memoriji na ovaj način, umesto korišćenja globalnih promenljivih, je to da samo funkcije sa pristupom pokazivaču imaju pristup podacima.

Stek i slobodno skladište: operatori new i delete

- NEW : memoriju na slobodnom skladištu u C ++ alocirate korišćenjem ključne reči new. Posle nje sledi tip objekta koji želite da alocirate tako da kompajler zna koliko se memorije zahteva. New unsigned short int alocira dva bajta na slobodnom skladištu, a new long alocira četiri. Povratna vrednost iz new je memorijska adresa. Ona se mora dodeliti pokazivaču. Da biste kreirali unsigned short na slobodnom skladištu, napišite:
-
- ***unsigned short int *pPointer;***
- ***pPointer = new unsigned short int;***
-
- U svakom slućaju, pPointer sada pokazuje na unsigned short int na slobodnom skladištu. Njega možete korisriti kao i svaki drugi pokazivač na promenljivu i dodeliti vednost području memorije.
- UPOZORINJE: Svaki put kada alocirate memoriju, korišćenjem ključne reći new, morate proveriti da biste bili sigurni da pokazivač nije nula.

Stek i slobodno skladište: operatori new i delete

- DELETE

- Kada završite sa Vašim područjem memorije, morare pozvati delete za pokazivač - on vraća memoriju slobodnom skladištu. Zapamtite da je sam pokazivač, što je suprotno memoriji na koju on pokazuje, lokalna promanljiva. Po povratku iz funkcije u kom je deklarisan, pokazivač izlazi iz opsega i postaje izgubljen. Memorija alocirana sa new se ne oslobađa automatski. Ona postaje neraspoloživa -situacija nazvana memorijska pukotina, jer se ta memorija ne može povratiti, dok se program ne završi. To je kao da je memorija "iscurela" iz vašeg kompjutera. Da biste vratili memoriju u slobodno skladište, koristite ključnu reč delete, naprimer:
- ***delete pPointer;***

- KADA BRIŠETE POKAZIVAČ, TO JE STVARNO OSLOBAĐANJE MEMORIJE ČIJA SE ADRESA ČUVA U POKAZIVAČU

Динамичка додела и ослобађање меморије

- **Оператор new**
- **додељује меморијски простор (ако је слободан) аутоматски, на основу броја објеката**
- **резултат примене оператора је адреса додељеног меморијског простора**
- **може иницијализовати додељен простор ако се не ради о низу променљивих**

- **Оператор delete**
- **ослобађа меморијски простор само ако је динамички додељен**

Динамичка додела и ослобађање меморије

```
adresa = new tip_promenljive;                    //za osnovni tip
adresa = new tip_promenljive(pocetna_vrednost);

adresa = new tip_niza[ velicina_niza];                  //za 1d niz
adresa = new tip_niza[ broj_nizova][velicina_niza];   //za 2d niz

adresa = new tip_strukture;                      //za strukturu
adresa = new tip_strukture[ velicina_niza];   //za 1d niz struktura

delete adresa;        //za pojedinacne promenljive i 1d nizove
delete [ ] adresa;    //samo za visedimenzionalne nizove,
                      //jedan par zagrada bez obzira na dimenzije!
```

Динамичка додела и ослобађање меморије

```
#include <iostream.h>

main()
{   int i, *niz;

    niz = new int[10];
    /*provera*/

    for(i=0; i<10; i++)
    {       cin >> niz[i];
            cout << niz[i] << ' ';      }
    cout << endl;

    delete niz;

    return 0;
}
```

Статичка декларација показивача

Динамичка додела за низ

Укидање низа

Динамичка додела и ослобађање меморије

```
#include <iostream.h>
#include <string.h>

struct s{ char ime[31]; int broj; };
void cita_pise( s niz[ ], int n);

main()
{   s *niz_s;
    niz_s = new s[100];
    /*provera*/

    cita_pise( niz_s, 100);

    delete niz_s;

    return 0;
}
```

```
void cita_pise( s niz[ ], int n)
{   int i;

    for(i=0; i<n; i++)
    {  cin >> niz[i].ime;
       cin >> niz[i].broj;  }

    for(i=0; i<n; i++)
    {  cout<<niz[i].ime<<' ';
       cout<<niz[i].broj<<' ';
       cout << endl;            }
}
```

Dynamic memory allocation

```
// Izračunavanje prostog broja korišćenjem dynamic memory allocation
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
int main()
{ long* pprime=0; // Pointer to prime array
long trial = 5; // Candidate prime
int count = 3; // Count of primes found
int found = 0; // Indicates when a prime is found
int max = 0; // Number of primes required
cout << endl
<< "Enter the number of primes you would like: ";
cin >> max; // Number of primes required
if(!(pprime=new long[max]))
{ cout << endl
<< "Memory allocation failed.";
exit(1); // Terminate program }
```

Dynamic memory allocation

- *pprime = 2; // Insert three
- *(pprime+1) = 3; // seed primes
- *(pprime+2) = 5;
- do
- { trial += 2; // Next value for checking
- found = 0; // Set found indicator
- for(int i = 0; i < count; i++)
- // Division by existing primes
- { found =(trial % *(pprime+i)) == 0; // True for exact division
- if(found) // If division is exact
- break; // it's not a prime }
- if (found == 0) // We got one... so save it in
- *(pprime+count++) = trial; // primes array }
- while(count < max);

Dynamic memory allocation

```
// Output primes 5 to a line
for(int i = 0; i < max; i++)
{
if(i%5 == 0) /* New line on 1st, and every 5th line*/
cout << endl;
cout << setw(10) << *(pprime+i); }
delete [ ] pprime; // Free up memory
cout << endl;
return 0;
}
```