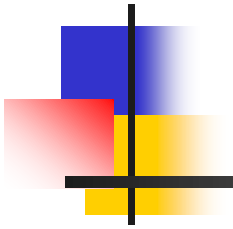
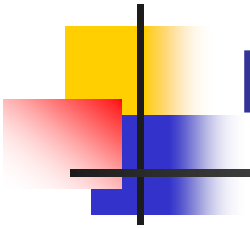


PROGRAMIRANJE I

P-03: Osnovi viših programskih jezika

prof. dr **Dražen Brđanin**
2023/24



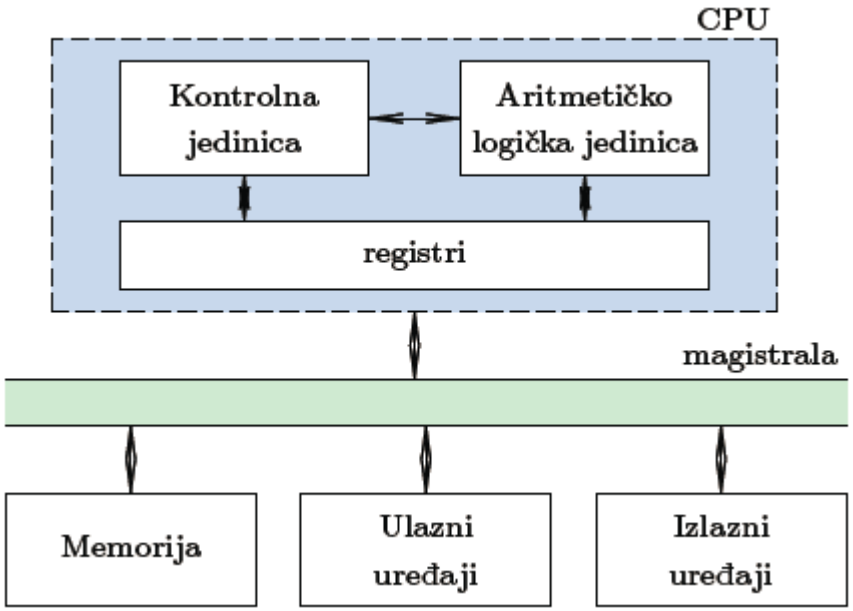
P-03: Osnovi viših programskih jezika

- **Sadržaj predavanja**
  - organizacija računarskog sistema
  - procedura razvoja aplikativnog softvera
  - programske paradigme
  - programski prevodioci
  - osnovni pojmovi o algoritmima

Organizacija računarskog sistema

računarski sistem = hardver + softver

hardver
(fizičke komponente računara)



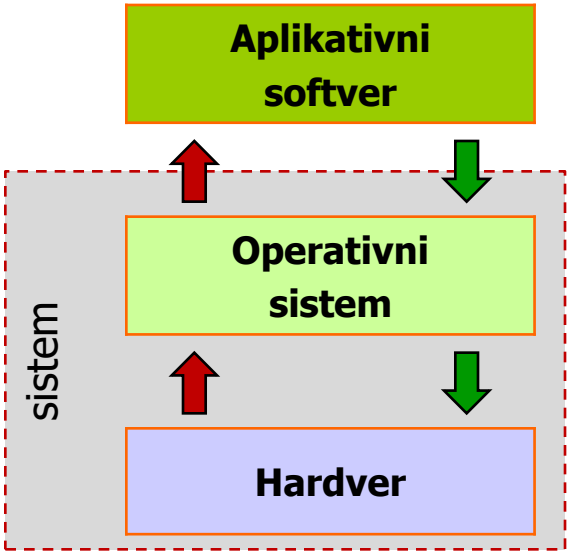
Fon Nojmanova arhitektura
(zajednička memorija za program i podatke)

softver
(programi i podaci)

Sistemski softver

operativni sistem +
uslužni softver (razvojni alati)

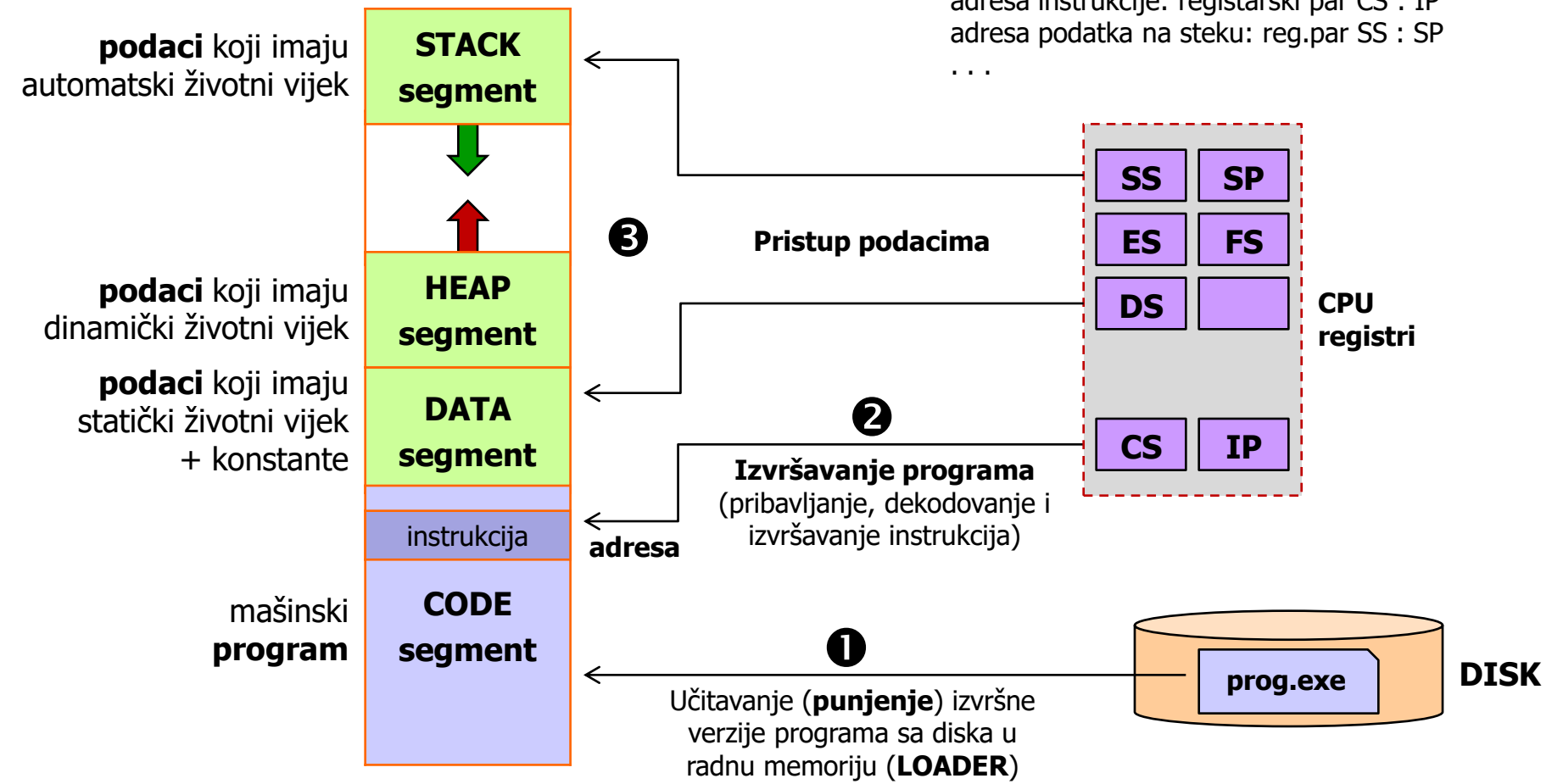
Aplikativni softver



Organizacija računarskog sistema

Fon Nojmanova arhitektura

(zajednička memorija za program i podatke)



Segmentna logička organizacija radne memorije

adresa podatka ili instrukcije određuje se odgovarajućim registarskim parom:

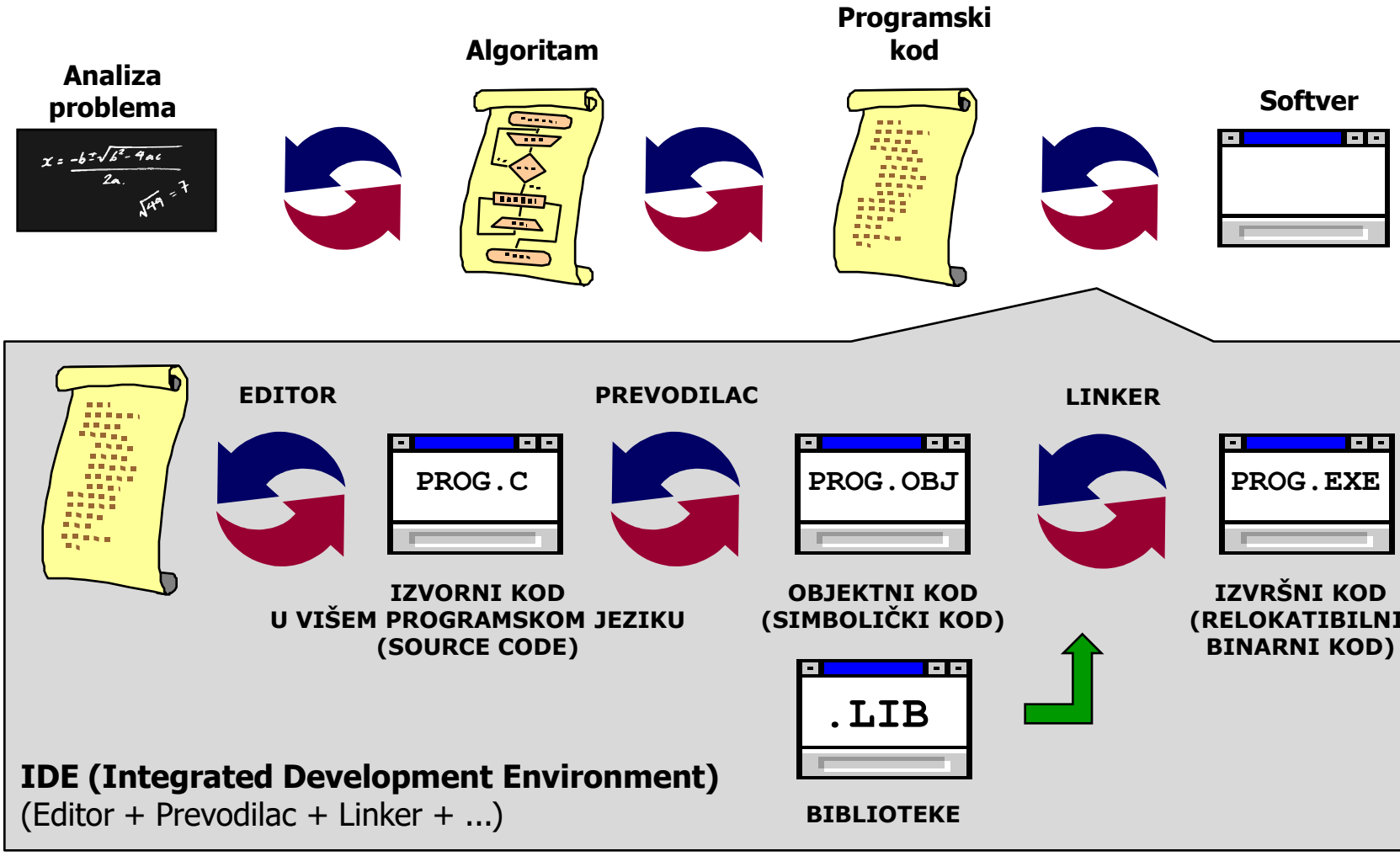
adresa instrukcije: registarski par CS : IP

adresa podatka na steku: reg.par SS : SP

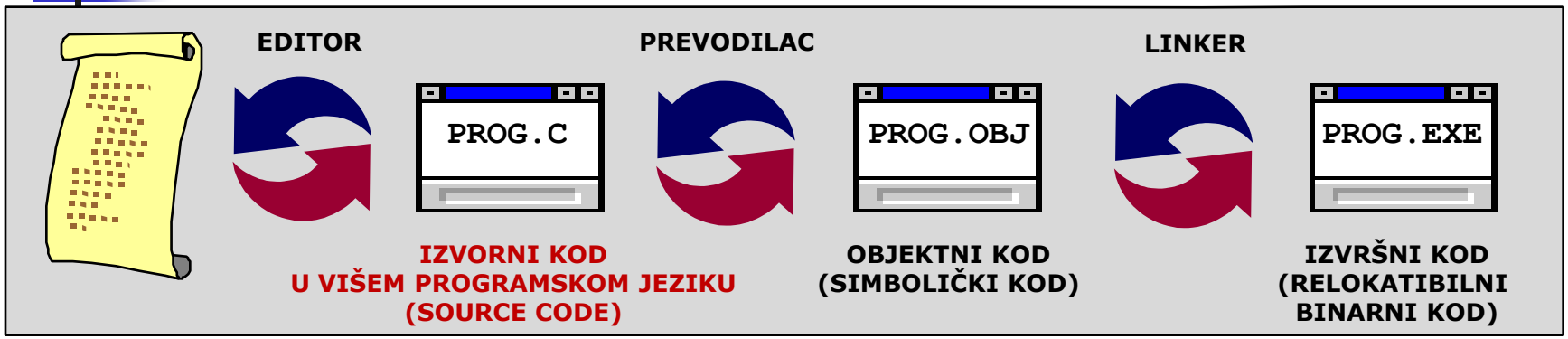
. . .

Procedura razvoja aplikativnog softvera

Razvojni ciklus programa u višim programskim jezicima



Osnovna podjela programskih jezika



Mašinski zavisni jezici

mašinski jezik

- **binarno kodovane instrukcije** koje procesor može direktno da izvršava

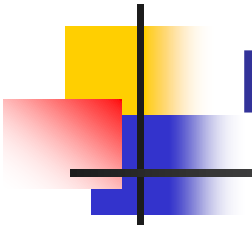
asemblerski (simbolički) jezik

- instrukcije nisu binarno kodovane – koriste se **mnemonici** – simboli (oznake) za pojedine mašinske instrukcije
- programiranje je jednostavnije, ali i dalje veoma blisko mašinskom jeziku
- **asembler** (prevodilac sa simboličkog na mašinski jezik)

Mašinski nezavisni jezici

viši programski jezici

- skrivaju detalje računarskog sistema i omogućavaju programiranje u jeziku nezavisnom od mašine
- prvi viši programski jezici nastaju 1950-ih godina
- **različite paradigme**
- jezici **opšte namjene** i **specijalizovani** jezici
- preko 8.000 programskih jezika!!!



Programske paradigme

Paradigma (grč.) = obrazac, šablon, ...

Programska paradigma

- **programski stil**
- obrazac programiranja
- **fundamentalni stil programiranja**

Povezanost paradigmi i programskih jezika

- **grupisanje/klasifikacija programskih jezika**
- broj paradigmi je daleko manji od broja jezika
- informacija da neki jezik pripada nekoj paradigmi govori o osnovnim svojstvima i mogućnostima jezika
- poznavanje određene paradigme značajno olakšava savladavanje jezika koji toj paradigmi pripada
- svakoj programskoj paradigmi pripada više jezika, npr:
  - proceduralna paradigma: Pascal, C, ...
  - objektno-orijentisana paradigma: Java, C++, ...
  - ...
- neki jezik može da podržava više paradigmi, npr: C++ podržava proceduralni stil, objektno-orijentisani stil, generički stil

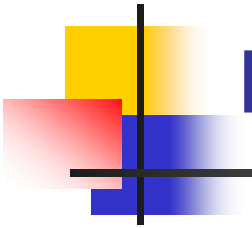
Najopštija podjela paradigmi

- **Proceduralna**

  osnovni zadatak programera da opiše **način** (proceduru) **kojim se dolazi do rješenja** problema

- **Deklarativna**

  osnovni zadatak programera je da **precizno opiše problem (šta želi)**, dok se mehanizam programskog jezika bavi pronalaženjem rješenja problema



Programske paradigme

Osnovne programske paradigme

Imperativna paradigma

- program specifikuje **proceduru** (redoslijed koraka) **za rješavanje problema**
- **osnovni element** imperativnih jezika je **naredba**
- naredbe se grupišu u procedure i izvršavaju se sekvencijalno ukoliko se eksplicitno u programu ne promijeni redoslijed izvršavanja
- osnovni elementi programa su **naredbe** (programska logika) i **promjenljive** (podaci)
- C, Pascal, ...

Funkcionalna paradigma

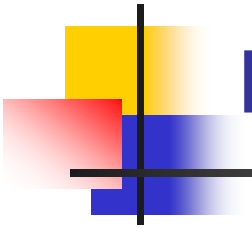
- nastala početkom 1960-ih godina, danas ponovo u usponu ...
- osnovni elementi su **matematičke funkcije** i ima formalnu strogo definisanu **matematičku osnovu u lambda računu**
- Lisp, Haskell, Scala, ...

Objektno-orijentisana paradigma

- osnovni elementi programa su **objekti**
- objekti su **instance klasa** – korisnički definisani tipovi podataka koji inkapsuliraju podatke i operacije
- klase su tipčno hijerarhijski organizovane i povezane mehanizmom nasljeđivanja
- aplikativna logika realizuje se interakcijom objekata, koji razmjenjuju poruke
- C++, Java, C#, ...

Logička paradigma

- deklarativna paradigma
- izvršavanje programa zasniva se na sistematskom pretraživanju skupa činjenica uz korištenje određenih pravila zaključivanja – zasnovano na matematičkoj logici (predikatski račun 1. reda)
- Prolog, Parlog, Solver, ...



Programske paradigme

Dodatne programske paradigme

Potparadigme osnovnih paradigmi ili kombinacija osnovnih paradigmi

Komponentna paradigma

- softver je kolekcija povezanih komponenata
- komponenta je kolekcija povezanih objekata koji obezbjeđuju neku funkcionalnost, koju druge komponente konzumiraju kroz određeni interfejs
- komponentno programiranje poželjno po principu "drag & drop"
- komponente mogu biti razvijene u različitim jezicima (Java, C++, ...)

Konkurentna paradigma

- istovremeno izvršavanje više programa
- **konkurentnost u užem smislu** (jedan procesor, zajednička memorija)
  - istovremeno izvršavanje više procesa u jednom procesoru uz korištenje zajedničke memorije
- **paralelno programiranje** (više procesora, zajednička memorija)
- **distribuirano programiranje** (više procesora, svaki sa svojom memorijom)

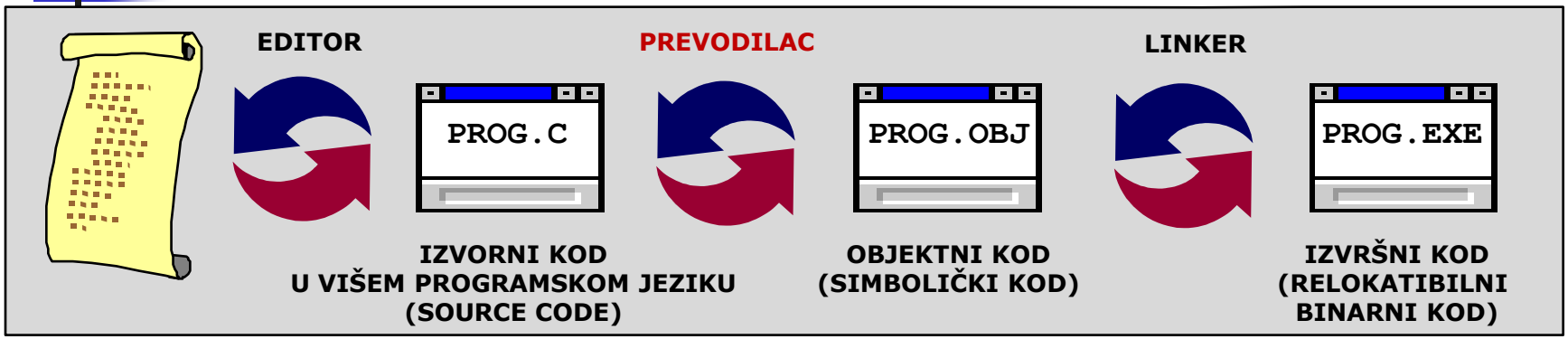
Paradigma upitnih jezika

- deklarativna paradigma
- **upitni jezici baza podataka** (SQL, Xquery, ...)
- ...

Reaktivna paradigma

- reaktivno programiranje usmjereno na tok podataka (izmjena jednog podatka utiče na drugi)
- **tabelarni proračuni (Excell),**
- **jezici za opis hardvera (Verilog), ...**

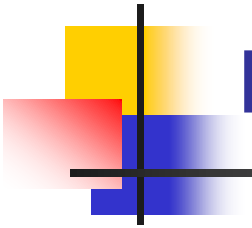
Programski prevodioci



Programski prevodioci (jezički procesori)

- Programi koji **analiziraju izvorni kod** u višem programskom jeziku i na osnovu ispravnog ulaznog programa **generišu odgovarajući mašinski kod**

- U zavisnosti od toga da li **se cijeli program analizira i transformiše u mašinski kod prije nego što može da se izvrši**, ili se **analiza i izvršavanje programa obavljaju naizmjenično dio po dio programa** (npr. naredba po naredba):
  - **kompilacija**
    - **kompilatori** (**kompajleri**)
  - **interpretacija**
    - **interpretatori** (**interpreteri**)



Programski prevodioci

Kompilatori (kompajleri)

- programski prevodioci kod kojih su faza prevođenja i faza izvršavanja programa potpuno razdvojene
- nakon analize izvornog koda programa u višem programskom jeziku, kompajleri generišu izvršni (mašinski) kod i dodatno ga optimizuju, a zatim čuvaju u izvršnim binarnim datotekama
- jednom sačuvani mašinski kod moguće je izvršavati neograničen broj puta, bez potrebe za ponovnim prevođenjem
- krajnjim korisnicima nije neophodno dostavljati izvorni kod programa na višem programskom jeziku, već je dovoljno distribuirati izvršni mašinski kod
- jedan od problema u radu sa kompajlerima je da se prevođenjem gubi svaka veza između izvršnog i izvornog koda – svaka (i najmanja) izmjena u izvornom kodu programa zahtijeva ponovno prevođenje programa ili njegovih dijelova
- kompajlirani programi su obično veoma efikasni

Interpretatori (interpreteri)

- programski prevodioci kod kojih su faza prevođenja i faza izvršavanja programa ispreplетane
- interpreteri analiziraju dio po dio (najčešće naredbu po naredbu) izvornog koda i odmah nakon analize vrše i njegovo izvršavanje
- rezultat prevođenja se ne smješta u izvršne datoteke, već je prilikom svakog izvršavanja neophodno iznova vršiti analizu izvornog koda
- programi koji se interpretiraju obično se izvršavaju znatno sporije nego u slučaju kompilacije
- razvojni ciklus programa je često kraći ukoliko se koriste interpreteri – prilikom malih izmjena programa nije potrebno iznova vršiti analizu cjelokupnog koda

Programski prevodioci

Kompilacija + Interpretacija

- za neke jezike postoje i interpreteri i kompajleri – intepreter se koristi u fazi razvoja programa, a u fazi eksploatacije koristi se kompajler koji proizvodi program sa efikasnim izvršavanjem
- danas se često primjenjuje i tehnika **kombinovanja kompilacije i interpretacije** –
  - prvo se kod sa višeg programskog jezika kompajlira u neki precizno definisan međujezik niskog nivoa (obično jezik neke apstraktne virtuelne mašine),
  - zatim se vrši interpretacija ovog međujezika i njegovo izvršavanje na konkretnom računaru
  - primjenjuje se kod jezika: Java, C#, ...

Proces prevođenja

- **leksička analiza** koda
  - analiza osnovnih gradivnih elemenata programa (lekseme)
- **sintaktička (sintaksna) analiza** koda
  - analiza ispravnosti jezičkih konstrukcija (iskazi)
- **semantička analiza** koda
  - analiza semantike (značenja) iskaza

Leksika + Sintaksa + Semantika

izučavaju se i za prirodne jezike, ne samo za vještačke (programske) jezike

+ Pragmatika

pragmatika se bavi izražajnošću jezika, stilovima, ...

Programski prevodioci

Leksika

- osnovni elementi prirodnog jezika su **riječi**
- postoje različite vrste riječi (imenice, glagoli, pridjevi, ...)
- riječi mogu da imaju različite oblike (padež, vrijeme, broj, ...)
- zadatak leksičke analiza prirodnih jezika jeste identifikacija i kategorizacija riječi u rečenicama
- zadatak leksičke analize programa napisanih u višem programskom jeziku jeste **identifikacija** i **kategorizacija leksičkih elemenata programa**
  - **lekseme (riječi)** – **leksički elementi**
  - **tokeni** – **kategorije**
- leksičku analizu vrši **leksički analizator** (komponenta prevodioca zadužena za leksičku analizu)

Primjer izvornog koda:

```
if ( x>2 )
    y = x+1 ;
```

Rezultat leksičke analize:

`if`	ključna riječ
`(`	zagrada
`x`	identifikator
`>`	operator
`2`	cjelobrojni literal
`)`	zagrada
`y`	identifikator
`=`	operator
`x`	identifikator
`+`	operator
`1`	cjelobrojni literal
`;`	punktuator

Programski prevodioci

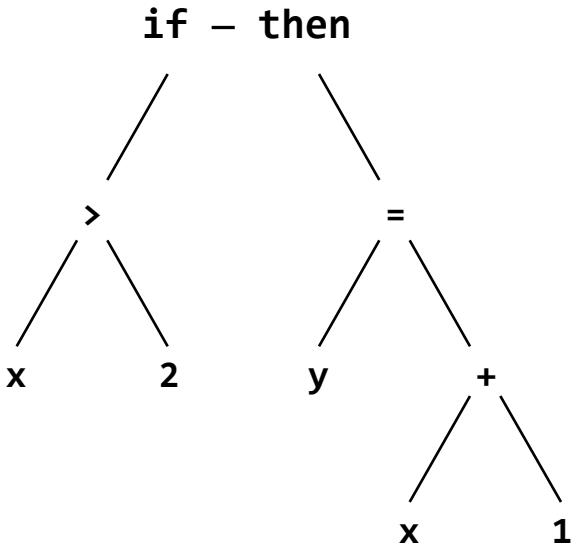
Sintaksa

- kod prirodnih jezika, **sintaksa definiše načine na koji pojedinačne riječi mogu da kreiraju ispravne rečenice**
- u programskim jezicima, **sintaksa definiše načine na koji se mogu formirati ispravni programi**, tj. **načine na koji se lekseme mogu grupisati u ispravne programske iskaze**
- **sintaksa definiše formalne relacije između elemenata jezika**, time pružajući strukturne opise ispravnih iskaza
- **sintaksa se bavi samo formom i strukturom jezika bez bilo kakvih razmatranja u vezi sa njihovim značenjem**
- **sintaksna struktura rečenica ili programa se može predstaviti u obliku stabla**

Primjer izvornog koda:

```
if ( x>2 )
    y = x++ ;
```

Sintaksno stablo:



Programski prevodioci

Semantika

- **semantika pridružuje značenje sintaksno ispravnim iskazima**
- npr, iskaz

```
if (x>2) y=x+1;
```

ima sljedeće značenje:

"ako je vrijednost promjenljive x veća od 2, tada promjenljivoj y dodijeli vrijednost zbira promjenljive x i broja 1"

- **nemaju svi sintaksno ispravni iskazi ispravno značenje,** npr. "bezbojna tečnost je plava"
- **neki apekti semantičke ispravnosti programa mogu da se provjere tokom prevođenja programa, a neki ne mogu,** npr.

```
y=1/x;
```

ako je x u toku izvršavanja jednako 0, imaćemo dijeljenje s nulom, što nije dozvoljeno u slučaju cijelih brojeva

- **statička semantika**
  - ponašanje programa unaprijed poznato i ne zavisi od drugih elemenata
- **dinamička semantika**
  - ponašanje programa nije unaprijed poznato i zavisi od drugih elemenata, npr. od vrijednosti neke promjenljive

- **Za neke programske jezike semantika je potpuno formalno definisana**
- **Semantika jezika C nije u potpunosti definisana standardom – moguće su različite interpretacije, u zavisnosti od implementacije**, npr.

```
f() + g()
```

nije definisano da li će se prvo izvršti funkcija f() ili funkcija g(), to zavisi od prevodioca

Strukturno programiranje – algoritmi

PROCEDURALNA / IMPERATIVNA PARADIGMA

- **PROGRAMER SPECIFIKUJE PROCEDURU / REDOSLIJED KORAKA / ALGORITAM ZA RJEŠAVANJE PROBLEMA**

ALGORITAM

- **tačno određen tok izvođenja nekog postupka koji se primjenjuje na nekom skupu podataka radi dobijanja rezultata, a koji se koristi pri rješavanju problema istog tipa**
- termin potiče od imena perzijskog matematičara koji je prvi dao pravila za izvođenje algebarskih operacija
- alternativni nazivi: RECEPT, PROCEDURA, ...

Osnovne karakteristike algoritma

- **Konačnost**
  - mora da vodi rješenju primjenom konačnog broja algoritamskih koraka
  - kad broj koraka nije unaprijed poznat, treba predvidjeti postupak koji ograničava broj koraka!
- **Definisanost**
  - svaki korak mora da bude jednoznačno definisan
  - treba predvidjeti sve slučajeve bez obzira na različite početne podatke
- **Ulaz(i)**
  - algoritam može da ima jedan ili više ulaznih podataka, ali i ne mora da ih ima!
  - mogu da se navedu na početku ili se dinamički unose
- **Izlaz(i)**
  - algoritam mora da ima jedan ili više izlaznih podataka
- **Efikasnost**
  - zahtijeva se postizanje traženog rješenja u što kraćem vremenu i uz primjenu što manjeg broja koraka
- **Ostvarljivost**
  - Algoritam mora biti ostvarljiv u računaru !!!

Struktura i reprezentacija algoritma

Struktura algoritma

- **Operatori**
  - uzastopnom primjenom operatora realizuje se željeni postupak
  - primjena pojedinog operatora predstavlja jedan algoritamski korak
  - u širem smislu, u skup operatora, pored operatora neohodnih za neposrednu realizaciju postupka, ubrajaju se i operacije neohodne za realizaciju algoritma u računaru (učitavanje podataka, ispisivanje rezultata, privremeni prekid obrade ...)
- **Diskriminatori**
  - tačke odlučivanja u kojima se na osnovu osobina operanada (podataka) mijenja redoslijed ili vrsta operatora

Reprezentacija algoritma

- **Prirodnim jezikom**
  - algoritamski koraci se opisuju prirodnim jezikom (jezik koji se koristi u svakodnevnoj komunikaciji)
  - nedostaci: nije univerzalan i prepoznatljiv...
- **Metajezikom (presudokod)**
  - metajezik je vještački jezik nezavisan od računarske platforme i neopterećen formalizmom kako bi se olakšao razvoj algoritma
  - često se koristi u literaturi za predstavljanje algoritama
  - univerzalniji je od prirodnog jezika
  - (uglavnom se koristi neki surogat engleskog jezika)
- **Grafički**
  - algoritamski koraci predstavljaju se odgovarajućim grafičkim simbolima
  - često se koristi **dijagram toka** (**blok dijagram** ili organigram)

Reprezentacija algoritma

Reprezentacija algoritma prirodnim jezikom

- algoritamski koraci se opisuju prirodnim jezikom (jezik koji se koristi u svakodnevnoj komunikaciji)
- nedostaci: nije univerzalan i prepoznatljiv...

Primjer:

Prirodnim jezikom opisati algoritam koji za proizvoljnu vrijednost poluprečnika kruga računa i ispisuje njegov obim i površinu.

```
1. POČETAK
2. UČITAJ POLUPREČNIK (r)
3. IZRAČUNAJ OBIM (O = 2*r*PI)
4. IZRAČUNAJ POVRŠINU (P = r*r*PI)
5. ISPIŠI OBIM (O)
6. ISPIŠI POVRŠINU (P)
7. KRAJ
```

Reprezentacija algoritma pseudokom

- vještački jezik nezavisan od računarske platforme i neopterećen formalizmom kako bi se olakšao razvoj algoritma
- često se koristi u literaturi za predstavljanje algoritama, univerzalniji je od prirodnog jezika (uglavnom neki surogat engleskog jezika)

Primjer:

Pseudokodom opisati algoritam koji za proizvoljnu vrijednost poluprečnika kruga računa i ispisuje njegov obim i površinu

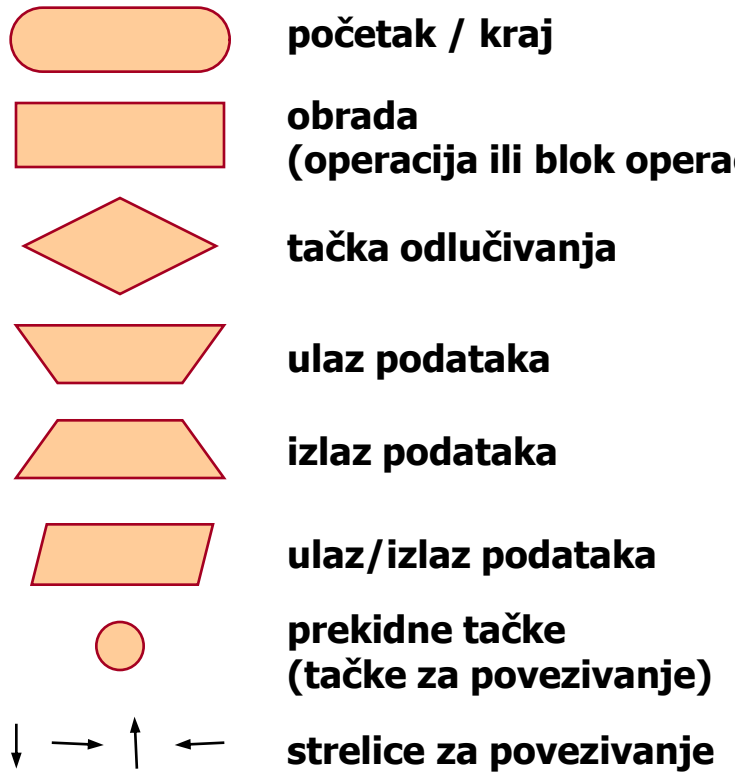
```
1. BEGIN
2. READ r
3. O = 2*r*PI
4. P = r*r*PI
5. WRITE O
6. WRITE P
7. END
```

Reprezentacija algoritma

Reprezentacija algoritma dijagramom toka

- algoritamski koraci predstavljaju se odgovarajućim grafičkim simbolima

Notacija dijagrama toka



početak / kraj

obrada
(operacija ili blok operacija)

tačka odlučivanja

ulaz podataka

izlaz podataka

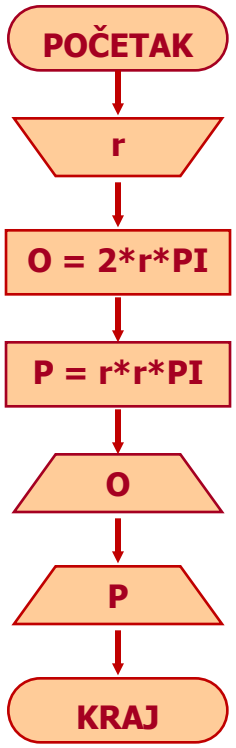
ulaz/izlaz podataka

prekidne tačke
(tačke za povezivanje)

strelice za povezivanje

Primjer :

Dijagramom toka predstaviti algoritam koji za proizvoljnu vrijednost poluprečnika kruga računa i ispisuje njegov obim i površinu

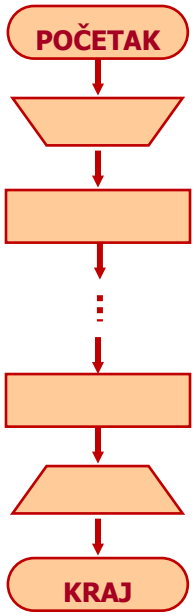



```
1. POČETAK (START, BEGIN)
2. UČITAJ POLUPREČNIK (r)
3. IZRAČUNAJ OBIM (O=2*r*PI)
4. IZRAČUNAJ POVRŠINU (P=r*r*PI)
5. ISPIŠI OBIM (O)
6. ISPIŠI POVRŠINU (P)
7. KRAJ
```

Linearna algoritamska struktura

Linearna algoritamska struktura

- linearna algoritamska struktura predstavlja **niz algoritamskih koraka (operatora) bez diskriminatora.**
- ovakva struktura izvršava se **sukcesivno** – **korak po korak, bez ponavljanja**
- od početka do kraja algoritma postoji **samo jedan mogući put** (**nema grananja**)
- to su algoritmi kod kojih obično imamo **ulaz, obradu i izlaz podataka**



Implementacija u višim prog. jezicima

- **algoritamski koraci reprezentuju se odgovarajućim programskim iskazima u skladu sa sintaksom programskog jezika (C: alg. korak = naredba izraza)**
- pored iskaza koji reprezentuju algoritamske korake (primjena operatora), program može da sadrži i druge iskaze, kao što su pretprocesorske direktive, deklaracije, ...
- **različiti jezici – različita pravila i različita struktura programa**

Primjer programa (C):

```
#include <stdio.h>
int main()
{
   printf("ETF\n");
   return 0;
}
```

Primjer programa (C++):

```
#include <iostream>
using namespace std;
int main()
{
   cout << "ETF" << endl;
   return 0;
}
```

Primjer programa (Pascal):

```
PROGRAM primjer;
USES crt;
BEGIN
   WRITELN('ETF');
END.
```