Uvodu u informatiku – Programski jezici i prevodioci

Danijela Simić
programski jezici i prevodioci

1. oktobar 2024.

Sadržaj

Uvod

Uvod

- Razvoj programskih jezika je bio u bliskoj vezi sa razvojem računara.

Uvod

- Razvoj programskih jezika je bio u bliskoj vezi sa razvojem računara.
- Prvi programski jezici zahtevali su od programera da bude upoznat sa najfinijim detaljima računara za koji se piše program.

Uvod

- Razvoj programskih jezika je bio u bliskoj vezi sa razvojem računara.
- Prvi programski jezici zahtevali su od programera da bude upoznat sa najfinijim detaljima računara za koji se piše program.
- Polovinom 1950-ih godina nastali su *viši programski jezici*.

Uvod

Da bi viši programski jezici mogli da se koriste postoje specijalizovani programi, tzv. jezički procesori ili programski prevodioci.

Uvod

Da bi viši programski jezici mogli da se koriste postoje specijalizovani programi, tzv. jezički procesori ili programski prevodioci.

Uvod

- Ne postoji *najbolji* programski jezik.

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.
- Indeks TIOBE

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.
- Indeks TIOBE
- C je dugo godina bio uvek na prvom ili drugom mestu

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.
- Indeks TIOBE
- C je dugo godina bio uvek na prvom ili drugom mestu
- od 2000. godine jezik Java je takođe dugo bio na prvom ili drugom, a C++ u prvih pet.

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.
- Indeks TIOBE
- C je dugo godina bio uvek na prvom ili drugom mestu
- od 2000. godine jezik Java je takođe dugo bio na prvom ili drugom, a C++ u prvih pet.
- Januar 2021: C, Java, Python, C++, C#, Visual Basic, JavaScript, PHP

Uvod

- Ne postoji *najbolji* programski jezik.
- Neki izvori navode da je u realnoj upotrebi bilo do sada oko 250 jezika.
- Ne postoji ni *najzastupljeniji* ili *najpopulariniji* programski jezik.
- Indeks TIOBE
- C je dugo godina bio uvek na prvom ili drugom mestu
- od 2000. godine jezik Java je takođe dugo bio na prvom ili drugom, a C++ u prvih pet.
- Januar 2021: C, Java, Python, C++, C#, Visual Basic, JavaScript, PHP
- Septembar 2024: Python, C++, Java, C, C#, JavaScript, Visual Basic, Go.

Faze razvoja softvera

- Pisanje: kreiranje *izvorni program* ili *izvorni kôd*.

Faze razvoja softvera

- Pisanje: kreiranje *izvorni program* ili *izvorni kôd*.
- Prevođenje: na asemblerski, mašinski jezik; *objektni kôd*.

Faze razvoja softvera

- Pisanje: kreiranje *izvorni program* ili *izvorni kôd*.
- Prevođenje: na asemblerski, mašinski jezik; *objektni kôd*.
- Povezivanje: *povezivanja* više objektnih programa, kreiranje izvršivog programa.

Klasifikacije programskih jezika

Klasifikacije programskih jezika

Najopštija podela viših programskih jezika je podela po načinu rešavanja problema.

Klasifikacije programskih jezika

Najopštija podela viših programskih jezika je podela po načinu rešavanja problema.

Po načinu rešavanja problema, programski jezici se dele na proceduralne i deklarativne.

Klasifikacije programskih jezika

Najopštija podela viših programskih jezika je podela po načinu rešavanja problema.

Po načinu rešavanja problema, programski jezici se dele na proceduralne i deklarativne.

Proceduralni jezici

Većina programskih jezika danas je *proceduralna* što znači da je zadatak programera da precizno opiše način (proceduru) kojim se dolazi do rešenja problema.

Proceduralni jezici

Većina programskih jezika danas je *proceduralna* što znači da je zadatak programera da precizno opiše način (proceduru) kojim se dolazi do rešenja problema.

```
cena1 = int(input('Unesi cenu prvog artikla: '))
cena2 = int(input('Unesi cenu drugog artikla: '))

if cena1 < cena2:
    print('Prvi artikal je jeftiniji')
else:
    print('Drugi artikal je jeftiniji')
```

Značajni proceduralni programski jezici su, na primer, C, Pascal, Python i Java.

Deklarativni jezici

Deklarativni jezici

Deklarativni programski jezici od programera zahtevaju da precizno opiše problem, dok se mehanizam programskog jezika onda bavi pronalaženjem rešenja problema.

Deklarativni jezici

Deklarativni jezici

Deklarativni programski jezici od programera zahtevaju da precizno opiše problem, dok se mehanizam programskog jezika onda bavi pronalaženjem rešenja problema.

- Olakšava proces programiranja.

Deklarativni jezici

Deklarativni jezici

Deklarativni programski jezici od programera zahtevaju da precizno opiše problem, dok se mehanizam programskog jezika onda bavi pronalaženjem rešenja problema.

- Olakšava proces programiranja.
- Ipak, često su neefikasni su pronalaženju efikasnih rešenja.

Deklarativni jezici

```
% Činjenice
roditelj(jovan, marija).          roditelj(jovan, mihajlo).
roditelj(suzana, marija).         roditelj(suzana, mihajlo).
roditelj(marija, ana).

% Pravila
otac(X, Y) :- roditelj(X, Y), musko(X).
majka(X, Y) :- roditelj(X, Y), zensko(X).
deda_baba(X, Y) :- roditelj(X, Z), roditelj(Z, Y).

% Činjenice o polu
musko(jovan).          musko(mihajlo).
zensko(suzana).        zensko(marija).
zensko(ana).
```

Klasifikacije programskih jezika

Programske paradigme

Programske paradigme

Programske paradigme predstavljaju različite stilove programiranja koji često služe i za klasifikaciju programskih jezika.

Razlike među paradigmama:

- konceptima i apstrakcijama (promenljive, funkcije, objekti, ograničenja);

Programske paradigme

Programske paradigme predstavljaju različite stilove programiranja koji često služe i za klasifikaciju programskih jezika.

Razlike među paradigmama:

- konceptima i apstrakcijama (promenljive, funkcije, objekti, ograničenja);
- koraci (dodele, sračunavanja vrednosti izraza, tokovi podataka, itd.).

Programske paradigme

- Izučavanje programskih paradigmi značajno olakšava razumevanje programskih jezika kao i očekivanja i mogućnosti koje jezik pruža.

Programske paradigme

- Izučavanje programskih paradigmi značajno olakšava razumevanje programskih jezika kao i očekivanja i mogućnosti koje jezik pruža.
- Poznavanje određene paradigme nam omogućava da brže i lakše savladamo svaki programski jezik koji toj paradigmi pripada.

Programske paradigme

- Izučavanje programskih paradigmi značajno olakšava razumevanje programskih jezika kao i očekivanja i mogućnosti koje jezik pruža.
- Poznavanje određene paradigme nam omogućava da brže i lakše savladamo svaki programski jezik koji toj paradigmi pripada.
- imperativno, funkcionalno, objektno-orjentisano i logičko programiranje.

Klasifikacije programskih jezika

Imperativni jezici

Imperativni jezici

Imperativni jezici

U ovim jezicima stanje programa karakterišu *promenljive* kojima se predstavljaju podaci i *naredbe* kojima se vrše određene transformacije promenljivih.

Imperativni jezici su obično izrazito proceduralni.

Imperativni jezici

Vrednosti promenljivih se menjaju naredbom dodele, a kontrola toka programa se vrši korišćenjem sekvence (nizanje naredbi), selekcije (izbor koja će naredba biti izvršena u zavisnosti od uspunjenosti nekog uslova) i iteracije (ponavljanje izvršavanja naredbi).

Imperativni jezici

Vrednosti promenljivih se menjaju naredbom dodele, a kontrola toka programa se vrši korišćenjem sekvence (nizanje naredbi), selekcije (izbor koja će naredba biti izvršena u zavisnosti od uspunjenosti nekog uslova) i iteracije (ponavljanje izvršavanja naredbi).

Imperativni jezici, uz objektno-orijentisane jezike, se najčešće koriste u industrijskom, sistemskom i aplikativnom programiranju.

Imperativni jezici

```
#include<stdio.h>
int main() {
  int cena1, cena2;

  printf("Unesi cenu prvog artikla\n");
  scanf("%d", &cena1);
  printf("Unesi cenu drugog artikla\n");
  scanf("%d", &cena2);

  if(cena1 < cena2)
      print("Prvi artikal je jeftiniji\n");
  else
      printf("Drugi artikal je jeftiniji\n");

  return 0;
```

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.
  - Početna verzija kompilatora za Fortran I objavljena je nekoliko godina kasnije – 1956. godine.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.
  - Početna verzija kompilatora za Fortran I objavljena je nekoliko godina kasnije – 1956. godine.
  - Znatno je olakšano i održavanje programa zbog bolje čitljivosti i omogućena je prenosivost između različitih računara

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.
  - Početna verzija kompilatora za Fortran I objavljena je nekoliko godina kasnije – 1956. godine.
  - Znatno je olakšano i održavanje programa zbog bolje čitljivosti i omogućena je prenosivost između različitih računara
  - Već 1958. više od polovine svih programa pisano je na Fortran-u.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.
  - Početna verzija kompilatora za Fortran I objavljena je nekoliko godina kasnije – 1956. godine.
  - Znatno je olakšano i održavanje programa zbog bolje čitljivosti i omogućena je prenosivost između različitih računara
  - Već 1958. više od polovine svih programa pisano je na Fortran-u.
  - Danas se koristi i namenjen je za numerička i naučna izračunavanja.

Značajni imperativni jezici

- Fortran – nastao u periodu 1953-1957; *FORmula TRANslating System*, Džon Bakus i IBM.
  - Prvi interpretator za Fortran bio je razvijen 1953. godine.
  - Programiranje je postalo brže, ali novi programi su se izvršavali 10-20 puta sporije nego programi napisani na asembleru.
  - Početna verzija kompilatora za Fortran I objavljena je nekoliko godina kasnije – 1956. godine.
  - Znatno je olakšano i održavanje programa zbog bolje čitljivosti i omogućena je prenosivost između različitih računara
  - Već 1958. više od polovine svih programa pisano je na Fortran-u.
  - Danas se koristi i namenjen je za numerička i naučna izračunavanja.
- Cobol (*COmmon Business Oriented Language*) – 1959. godine; za izradu poslovnih, finansijskih i administrativnih aplikacija.

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.
- Pascal – jedan od naslednika jezika Algol, nastao 1970. godine; ohrabruje korišćenje strukturiranog programiranja; smatrano da je njegovo glavno polje nastava programiranja.

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.
- Pascal – jedan od naslednika jezika Algol, nastao 1970. godine; ohrabruje korišćenje strukturiranog programiranja; smatrano da je njegovo glavno polje nastava programiranja.
- Basic – razvijen 1964. godine; *Beginner's All-Purpose Symbolic Instruction Code*

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.
- Pascal – jedan od naslednika jezika Algol, nastao 1970. godine; ohrabruje korišćenje strukturiranog programiranja; smatrano da je njegovo glavno polje nastava programiranja.
- Basic – razvijen 1964. godine; *Beginner's All-Purpose Symbolic Instruction Code*
  - Za početnike u programiranju.

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.
- Pascal – jedan od naslednika jezika Algol, nastao 1970. godine; ohrabruje korišćenje strukturiranog programiranja; smatrano da je njegovo glavno polje nastava programiranja.
- Basic – razvijen 1964. godine; *Beginner's All-Purpose Symbolic Instruction Code*
  - Za početnike u programiranju.
  - Popularnost jezika porasla je sa pojavom mikro-računara i personalnih računara.

Značajni imperativni jezici

- Algol („ALGOrithmic Language“) – jedan od najuticajnijih programskih jezika.
- Pascal – jedan od naslednika jezika Algol, nastao 1970. godine; ohrabruje korišćenje strukturiranog programiranja; smatrano da je njegovo glavno polje nastava programiranja.
- Basic – razvijen 1964. godine; *Beginner's All-Purpose Symbolic Instruction Code*
  - Za početnike u programiranju.
  - Popularnost jezika porasla je sa pojavom mikro-računara i personalnih računara.
  - Rayvijeno nekoliko modifikacija i proširenja – Visual Basic.

Značajni imperativni jezici

- C – 1972. godine razvio Denis Riči.

Značajni imperativni jezici

- C – 1972. godine razvio Denis Riči.
  - Naslednik jezika B.

Značajni imperativni jezici

- C – 1972. godine razvio Denis Riči.
  - Naslednik jezika B.
  - Namenjen prevashodno pisanju sistemskog softvera u okviru sistema Unix.

Značajni imperativni jezici

- C – 1972. godine razvio Denis Riči.
  - Naslednik jezika B.
  - Namenjen prevashodno pisanju sistemskog softvera u okviru sistema Unix.
  - Počeo da se koristi i za pisanje aplikativnog softvera na velikom broju drugih platformi — od mikrokontrolera do superračunara.

Značajni imperativni jezici

- C – 1972. godine razvio Denis Riči.
  - Naslednik jezika B.
  - Namenjen prevashodno pisanju sistemskog softvera u okviru sistema Unix.
  - Počeo da se koristi i za pisanje aplikativnog softvera na velikom broju drugih platformi — od mikrokontrolera do superračunara.
  - Značajno je uticao i na razvoj drugih programskih jezika.

Klasifikacije programskih jezika

Funkcionalni jezici

Funkcionalni jezici

Funkcionalni jezici razmatraju programiranje kao proces izračunavanja matematičkih funkcija.

Funkcionalni jezici

Funkcionalni jezici razmatraju programiranje kao proces izračunavanja matematičkih funkcija.

- Koreni funkcionalnog programiranja leže u λ-računu.

Funkcionalni jezici

Funkcionalni jezici razmatraju programiranje kao proces izračunavanja matematičkih funkcija.

- Koreni funkcionalnog programiranja leže u λ-računu.
- Mnogi funkcionalni programski jezici mogu se smatrati nadogradnjama λ-računa.

Funkcionalni jezici

Funkcionalni jezici razmatraju programiranje kao proces izračunavanja matematičkih funkcija.

- Koreni funkcionalnog programiranja leže u λ-računu.
- Mnogi funkcionalni programski jezici mogu se smatrati nadogradnjama λ-računa.
- Funkcionalno programiranje je u ekspanziji.

Funkcionalni jezici

Funkcionalni jezici razmatraju programiranje kao proces izračunavanja matematičkih funkcija.

- Koreni funkcionalnog programiranja leže u λ-računu.
- Mnogi funkcionalni programski jezici mogu se smatrati nadogradnjama λ-računa.
- Funkcionalno programiranje je u ekspanziji.
- Neki programeri ih svrstavaju u deklarativnu paradigmu.

Suma kvadrata svih neparnih prirodnih brojeva čiji je kvadrat manji od 10000.

```
sum (takeWhile (<10000) (filter odd (map (^2) [1..])))
```

Značajni funkcionalni jezici

- Lisp – 1958. godine; *LISt Processing*; dugo smatran jezikom veštačke inteligencije.

Značajni funkcionalni jezici

- Lisp – 1958. godine; *LISt Processing*; dugo smatran jezikom veštačke inteligencije.
- Haskell – 1990. godine; često koristi u akademskim krugovima, ali ima i široku primenu u industriji za rešavanje složenih problema, posebno u oblastima gde su ispravnost i pouzdanost koda ključne.

Značajni funkcionalni jezici

- Lisp – 1958. godine; *LISt Processing*; dugo smatran jezikom veštačke inteligencije.
- Haskell – 1990. godine; često koristi u akademskim krugovima, ali ima i široku primenu u industriji za rešavanje složenih problema, posebno u oblastima gde su ispravnost i pouzdanost koda ključne.
- Erlang – 1980-ih u Ericssonu.

Značajni funkcionalni jezici

- Lisp – 1958. godine; *LISt Processing*; dugo smatran jezikom veštačke inteligencije.
- Haskell – 1990. godine; često koristi u akademskim krugovima, ali ima i široku primenu u industriji za rešavanje složenih problema, posebno u oblastima gde su ispravnost i pouzdanost koda ključne.
- Erlang – 1980-ih u Ericssonu.
  - Cilj da olakša izgradnju distribuiranih, skalabilnih i visoko dostupnih sistema.

Značajni funkcionalni jezici

- Lisp – 1958. godine; *LISt Processing*; dugo smatran jezikom veštačke inteligencije.
- Haskell – 1990. godine; često koristi u akademskim krugovima, ali ima i široku primenu u industriji za rešavanje složenih problema, posebno u oblastima gde su ispravnost i pouzdanost koda ključne.
- Erlang – 1980-ih u Ericssonu.
  - Cilj da olakša izgradnju distribuiranih, skalabilnih i visoko dostupnih sistema.
  - Koristi u raznim aplikacijama, kao što su telekomunikacioni sistemi, bankarstvo, i masivne online igre.

Značajni funkcionalni jezici

- Elixir – 2011. godina; naslednik Erlanga; popularan u razvoju veb aplikacija, odličan izbor za distribuirane sisteme koji zahtevaju visoku dostupnost, paralelno izvršavanje procesa.

Značajni funkcionalni jezici

- Elixir – 2011. godina; naslednik Erlanga; popularan u razvoju veb aplikacija, odličan izbor za distribuirane sisteme koji zahtevaju visoku dostupnost, paralelno izvršavanje procesa.
- Elm – nastao 2012. godine i napravljen je za razvoj korisničkih interfejsa na vebu. Koristi se često u kombinaciji sa jezikom Elixir, u kojem se programira logika sistema, dok se u Elmu programira korisnički interfejs.

Značajni funkcionalni jezici

- Elixir – 2011. godina; naslednik Erlanga; popularan u razvoju veb aplikacija, odličan izbor za distribuirane sisteme koji zahtevaju visoku dostupnost, paralelno izvršavanje procesa.
- Elm – nastao 2012. godine i napravljen je za razvoj korisničkih interfejsa na vebu. Koristi se često u kombinaciji sa jezikom Elixir, u kojem se programira logika sistema, dok se u Elmu programira korisnički interfejs.
- Clojure – nastao 2007. godine; naslednik Lisp; koristi se u kombinaciji sa Javom; za konkurentno i paralelno programiranje.

Klasifikacije programskih jezika

Logički jezici

Logički jezici

Logička paradigma je deklartivna paradigma koja se oslanja na matematičku logiku (konkretno, na metod rezolucije).

Logički jezici

Logička paradigma je deklartivna paradigma koja se oslanja na matematičku logiku (konkretno, na metod rezolucije).

Osnovni predstavnik ove paradigme je programski jezik Prolog.

Logički jezici – primer

Postoje tri kuće u nizu, svaka je obojena različitom bojom: crvena, plava i zelena. Svaki vlasnik kuće pije različito piće: vodu, čaj i kafu. Svaki vlasnik ima drugačiju životinju za kućnog ljubimca: mačku, psa i pticu.

Tragovi:

Osoba u crvenoj kući ima mačku. Osoba u zelenoj kući pije kafu. Osoba u plavoj kući pije čaj. Osoba u zelenoj kući nema pticu.

Koristeći tragove, rešiti zagonetku: Ko poseduje psa?

Logički jezici – primer

```
% Tri kuće: Kuca1, Kuca2, Kuca3
Kuce = [Kuca1, Kuca2, Kuca3],

% Svaka kuća je predstavljena listom [Boja, Pice, Ljubimac]
% Inicijalizujemo promenljive za boje, pića i ljubimce
Kuca1 = [Boja1, Pice1, Ljubimac1],
Kuca2 = [Boja2, Pice2, Ljubimac2],
Kuca3 = [Boja3, Pice3, Ljubimac3],

% Postoje tri moguće boje: crvena, zelena i plava
Boje = [crvena, zelena, plava],
% Postoje tri moguća pića: voda, čaj, kafa
Pica = [voda, caj, kafa],
```

Logički jezici – primer

```
% Postoje tri moguća ljubimca: mačka, pas, ptica
Ljubimci = [macka, pas, ptica],

% Svaka kuća ima različitu boju, piće i ljubimca
permutation(Boje, [Boja1, Boja2, Boja3]),
permutation(Pica, [Pice1, Pice2, Pice3]),
permutation(Ljubimci, [Ljubimac1, Ljubimac2, Ljubimac3]),


% Trag 1: Osoba u crvenoj kući ima mačku.
member([crvena, _, macka], Kuce),
```

Logički jezici – primer

```
% Trag 2: Osoba u zelenoj kući pije kafu.
member([zelena, kafa, _], Kuce),

% Trag 3: Osoba u plavoj kući pije čaj.
member([plava, caj, _], Kuce),

% Trag 4: Osoba u zelenoj kući nema pticu.
not(member([zelena, _, ptica], Kuce)),

% Određivanje ko ima psa i ispis rezultata
member([Boja, Pice, pas], Kuce), % Pronađi kuću sa psom
format('Osoba koja ima psa živi u kući koja je ~w
        i pije pice ~w.~n', [Boja, Pice]).
```

Klasifikacije programskih jezika

Objektno-orijentisani jezici

Objektno-orijentisani jezici

Objekti su specijalizovane strukture podataka koje uz polja podataka sadrže i metode kojima se manipuliše tim podacima.

Objektno-orijentisani jezici

Objekti su specijalizovane strukture podataka koje uz polja podataka sadrže i metode kojima se manipuliše tim podacima.

- Podaci se mogu obrađivati isključivo primenom metoda.

Objektno-orijentisani jezici

Objekti su specijalizovane strukture podataka koje uz polja podataka sadrže i metode kojima se manipuliše tim podacima.

- Podaci se mogu obrađivati isključivo primenom metoda.
- Značajniji objektno-orijentisani jezici su C++, Java i C#.

Objektno-orijentisani jezici

Najčešće korišćene tehnike programiranja u objektno orijentisanom programiranju uključuju sakrivanje informacija, enkapsulaciju, apstraktne tipove podataka, modularnost, nasleđivanje i polimorfizam.

Objektno-orijentisani jezici - primer Java

```
class Osoba {
    private String ime;
    private int godine;

    // Konstruktor
    public Osoba(String ime, int godine) {
        this.ime = ime;
        this.godine = godine;
    }

    // Metod za pozdrav
    public void predstaviSe() {
        System.out.println("Zdravo, ja sam " + ime +
        " i imam " + godine + " godina.");
    }
```

Objektno-orijentisani jezici - primer Java

```
    public static void main(String[] args) {
        // Kreiranje objekata
        Osoba osoba1 = new Osoba("Ana", 30);
        Osoba osoba2 = new Osoba("Marko", 25);

        // Pozivanje metoda za predstavljanje
        osoba1.predstaviSe();
        osoba2.predstaviSe();
    }
}
```

Značajni objektno-orijentisani jezici

- C++ – 1986. godine

Značajni objektno-orijentisani jezici

- C++ – 1986. godine
  - direktni naslednik jezika C;

Značajni objektno-orijentisani jezici

- C++ – 1986. godine
  - direktni naslednik jezika C;
  - jedan od najpopularnijih jezika;

Značajni objektno-orijentisani jezici

- C++ – 1986. godine
  - direktni naslednik jezika C;
  - jedan od najpopularnijih jezika;
  - bliske veza sa mašinom

Značajni objektno-orijentisani jezici

- C++ – 1986. godine
  - direktni naslednik jezika C;
  - jedan od najpopularnijih jezika;
  - bliske veza sa mašinom
  - razvoj zahtevnih aplikacija.

Značajni objektno-orijentisani jezici

- Java – 1995. godine

Značajni objektno-orijentisani jezici

- Java – 1995. godine
  - sintaksa jezika Java slična je jezicima C i C++ ali ima manje operacija niskog nivoa;

Značajni objektno-orijentisani jezici

- Java – 1995. godine
  - sintaksa jezika Java slična je jezicima C i C++ ali ima manje operacija niskog nivoa;
  - omogućava modifikacije koda u fazi izvršavanja;

Značajni objektno-orijentisani jezici

- Java – 1995. godine
  - sintaksa jezika Java slična je jezicima C i C++ ali ima manje operacija niskog nivoa;
  - omogućava modifikacije koda u fazi izvršavanja;
  - trenutno jedan od najpopularnijih programskih jezika;

Značajni objektno-orijentisani jezici

- Java – 1995. godine
  - sintaksa jezika Java slična je jezicima C i C++ ali ima manje operacija niskog nivoa;
  - omogućava modifikacije koda u fazi izvršavanja;
  - trenutno jedan od najpopularnijih programskih jezika;
  - kompilirani Java kôd (takozvani bajtkod) može izvršavati na bilo kojoj platformi koja podržava Javu (tj. koja raspolaže Java virtuelnom mašinom) bez ponovnog kompiliranja.

Značajni objektno-orijentisani jezici

C# – 2000. godine

- Micorosoft (u okviru .NET inicijative);

Značajni objektno-orijentisani jezici

C# – 2000. godine

- Micorosoft (u okviru .NET inicijative);
- po svojim karakteristikama je donekle sličan programskom jeziku Java;

Značajni objektno-orijentisani jezici

C# – 2000. godine

- Micorosoft (u okviru .NET inicijative);
- po svojim karakteristikama je donekle sličan programskom jeziku Java;
- jezik se stalno obogaćuje i unapređuje;

Značajni objektno-orijentisani jezici

C# – 2000. godine

- Micorosoft (u okviru .NET inicijative);
- po svojim karakteristikama je donekle sličan programskom jeziku Java;
- jezik se stalno obogaćuje i unapređuje;
- veoma popularan izbor za programiranje aplikacija za Windows i veb aplikacije.

Značajni objektno-orijentisani jezici

- Objective C i Swift – 1980-ih godina

Značajni objektno-orijentisani jezici

- Objective C i Swift – 1980-ih godina
  - koriste za razvoj aplikacija na platformama kompanije *Apple*;

Značajni objektno-orijentisani jezici

- Objective C i Swift – 1980-ih godina
  - koriste za razvoj aplikacija na platformama kompanije *Apple*;
  - Swift je danas primarni jezik za razvoj aplikacija na *Apple* platformama.

Klasifikacije programskih jezika

Savremene programske paradigme

Savremene programske paradigme

Savremeni programski jezici su multiparadigmatski, odnosno u sebi sadrže više različitih stilova programiranja.

- skript,
- komponentno,
- generičko,
- konkurentno i vizuelno,
- paradigme upitnih jezika i
- programiranja ograničenja

Skript

Skript programiranje je oblik programiranja koji se koristi za pisanje kratkih, jednostavnih programa ili „skripti“ koje automatizuju zadatke ili upravljaju funkcijama u većim softverskim sistemima.

Skript

Skript programiranje je oblik programiranja koji se koristi za pisanje kratkih, jednostavnih programa ili „skripti“ koje automatizuju zadatke ili upravljaju funkcijama u većim softverskim sistemima.

- Korisno za zadatke u kojima je važna brzina razvoja, fleksibilnost i jednostavnost.

Skript

Skript programiranje je oblik programiranja koji se koristi za pisanje kratkih, jednostavnih programa ili „skripti“ koje automatizuju zadatke ili upravljaju funkcijama u većim softverskim sistemima.

- Korisno za zadatke u kojima je važna brzina razvoja, fleksibilnost i jednostavnost.
- Skript jezici se koriste u domenu veb programiranja, skript jezici opšte namene, skript jezici za procesiranje teksta, komandni jezici id...

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;
  - ima elemente objektno-orijentisanih, imperativnih, funkcionalnih jezika;

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;
  - ima elemente objektno-orijentisanih, imperativnih, funkcionalnih jezika;
  - raspolaže ugrađenim strukturama podataka visokog nivoa;

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;
  - ima elemente objektno-orijentisanih, imperativnih, funkcionalnih jezika;
  - raspolaže ugrađenim strukturama podataka visokog nivoa;
  - programi interpretiraju, osnovni ciklus razvoja programa (pisanje-testiranje-debagovanje) odvija se izuzetno brzo;

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;
  - ima elemente objektno-orijentisanih, imperativnih, funkcionalnih jezika;
  - raspolaže ugrađenim strukturama podataka visokog nivoa;
  - programi interpretiraju, osnovni ciklus razvoja programa (pisanje-testiranje-debagovanje) odvija se izuzetno brzo;
  - najpopularniji jezik u oblastima kao što su istraživanje podataka i mašinsko učenje

Značajni skript jezici

- Perl – 1980-ih; moćne funkcije za obradu teksta, a koristio se u radu sa bazama podataka, u mrežnom programiranju i slično, ali i kao skript jezik za Linux sisteme.
- Python – 1991. godine
  - multiparadigmatski jezik;
  - jednostavna i izražajna sintakse;
  - ima elemente objektno-orijentisanih, imperativnih, funkcionalnih jezika;
  - raspolaže ugrađenim strukturama podataka visokog nivoa;
  - programi interpretiraju, osnovni ciklus razvoja programa (pisanje-testiranje-debagovanje) odvija se izuzetno brzo;
  - najpopularniji jezik u oblastima kao što su istraživanje podataka i mašinsko učenje
  - koristi i za veb-aplikacije, za mobilne aplikacije, za ugrađene sisteme i u nastavi...

Značajni skript jezici

- PHP – 1994. godina

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;
  - izvršava na serveru i dinamički generiše HTML sadržaj koji se šalje i prikazuje klijentu;

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;
  - izvršava na serveru i dinamički generiše HTML sadržaj koji se šalje i prikazuje klijentu;
  - više od polovine svih veb sajtova kao jezik na strani servera koriste PHP u nekom obliku.

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;
  - izvršava na serveru i dinamički generiše HTML sadržaj koji se šalje i prikazuje klijentu;
  - više od polovine svih veb sajtova kao jezik na strani servera koriste PHP u nekom obliku.
- JavaScript – 1995. godine

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;
  - izvršava na serveru i dinamički generiše HTML sadržaj koji se šalje i prikazuje klijentu;
  - više od polovine svih veb sajtova kao jezik na strani servera koriste PHP u nekom obliku.
- JavaScript – 1995. godine
  - koristi za programe koji se izvršavaju na strani klijenta (na primer, u okviru pregledača veba);

Značajni skript jezici

- PHP – 1994. godina
  - internet jezik koji može biti ugrađen u HTML kôd;
  - izvršava na serveru i dinamički generiše HTML sadržaj koji se šalje i prikazuje klijentu;
  - više od polovine svih veb sajtova kao jezik na strani servera koriste PHP u nekom obliku.
- JavaScript – 1995. godine
  - koristi za programe koji se izvršavaju na strani klijenta (na primer, u okviru pregledača veba);
  - omogućava da se sadržaj veb strana menja interaktivno.

Značajni skript jezici

- ASP.NET – 2002. godina

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;
  - dosta sličnosti sa jezikom PHP, ali može da se izvršava samo na Windows serverima.

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;
  - dosta sličnosti sa jezikom PHP, ali može da se izvršava samo na Windows serverima.
- Lua – 1993. godina

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;
  - dosta sličnosti sa jezikom PHP, ali može da se izvršava samo na Windows serverima.
- Lua – 1993. godina
  - jednostavna sintaksa;

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;
  - dosta sličnosti sa jezikom PHP, ali može da se izvršava samo na Windows serverima.
- Lua – 1993. godina
  - jednostavna sintaksa;
  - naročito primena u razvoju video igara;

Značajni skript jezici

- ASP.NET – 2002. godina
  - skript jezik za veb aplikacije i dinamičko kreiranje veb sadržaja kompanije Microsoft;
  - dosta sličnosti sa jezikom PHP, ali može da se izvršava samo na Windows serverima.
- Lua – 1993. godina
  - jednostavna sintaksa;
  - naročito primena u razvoju video igara;
  - podržava proceduralno, objektno orijentisano i funkcionalno programiranje, kao i programiranje vođeno podacima.

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.
- Bash – 1987. godine u okviru GNU projekta

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.
- Bash – 1987. godine u okviru GNU projekta
  - skriptni jezik za Unix i Unix-olike operativne sisteme, kao što su Linux i macOS;

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.
- Bash – 1987. godine u okviru GNU projekta
  - skriptni jezik za Unix i Unix-olike operativne sisteme, kao što su Linux i macOS;
  - koristi se za pisanje skripti koje mogu automatizovati razne zadatke, kao što su instalacija softvera, upravljanje datotekama i izvođenje sistemskih operacija;

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.
- Bash – 1987. godine u okviru GNU projekta
  - skriptni jezik za Unix i Unix-olike operativne sisteme, kao što su Linux i macOS;
  - koristi se za pisanje skripti koje mogu automatizovati razne zadatke, kao što su instalacija softvera, upravljanje datotekama i izvođenje sistemskih operacija;
  - često korsiti za administraciju sistema i upravljanje serverima.

Značajni skript jezici

- Ruby – 1995. godine; otovorenog je koda, sa fokusom na jednostavnost i produktivnost; osnovna ideja dizajna jezika je da se adresiraju ljudske potrebe.
- Bash – 1987. godine u okviru GNU projekta
  - skriptni jezik za Unix i Unix-olike operativne sisteme, kao što su Linux i macOS;
  - koristi se za pisanje skripti koje mogu automatizovati razne zadatke, kao što su instalacija softvera, upravljanje datotekama i izvođenje sistemskih operacija;
  - često korsiti za administraciju sistema i upravljanje serverima.
- R i S – 1976. godine i 1995. godine; specifične namene razvijeni za statističku analizu, vizualizaciju podataka i naučno istraživanje; R je veoma popularan u akademskim i istraživačkim krugovima, kao i u industrijama koje se bave analizom podataka.

Skript jezici – primer

Programski jezik Perl

```
print "Hello, World!\n";
```

Programski jezik Python

```
print("Hello, World!")
```

Programski jezik PHP

```
<?php
echo "Hello, World!";
```

Skript jezici – primer

Programski jezik JavaScript

```
console.log("Hello, World!");
```

Programski jezik Lua

```
print("Hello, World!")
```

Programski jezik Ruby

```
puts "Hello, World!"
```

Programski jezik Bash

```
echo "Hello, World!"
```

Konkurentno programiranje

Konkurentno programiranje je pristup pisanju koda koji omogućava izvršavanje više zadataka u istom vremenskom intervalu (bilo istovremeno ili vremenski isprepletano).

Konkurentno programiranje

Konkurentno programiranje je pristup pisanju koda koji omogućava izvršavanje više zadataka u istom vremenskom intervalu (bilo istovremeno ili vremenski isprepletano).

Ovaj pristup je ključan u razvoju modernih softverskih sistema, posebno u aplikacijama koje obrađuju velike količine podataka, zahtevaju visoku dostupnost ili performanse.

Konkurentno programiranje – značajni jezici

- Go, Golang – 2007. godine, Google; veoma popularan, naročito za razvoj servera, mrežnog softvera i za distribuiranu obradu podataka.

Konkurentno programiranje – značajni jezici

- Go, Golang – 2007. godine, Google; veoma popularan, naročito za razvoj servera, mrežnog softvera i za distribuiranu obradu podataka.
- Rust – 2010. godina, Mozilla; bezbedna i brza alternativa jezicima poput C i C++ , sa posebnim fokusom na onemogućavanje grešaka u radu sa memorijom; koristi se za veb servere, igre, alate komandne linije i komponente operativnih sistema kao što je Linux kernel.

Konkurentno programiranje – značajni jezici

Kotlin – 2011. godine; moderan programski jezik koji se izvršava na Java virtuelnoj mašini i koristi se primarno za razvoj Android aplikacija.

Konkurentno programiranje – primer

```
package main
import (
    "fmt"
    "sync")

// Funkcija za sabiranje elemenata dela niza
func sumPart(arr []int, result *int, wg *sync.WaitGroup)
    defer wg.Done()
    partialSum := 0
    for _, v := range arr {
        partialSum += v
    }
    *result = partialSum
}
```

Konkurentno programiranje – primer

```
func main() {
    arr := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    var wg sync.WaitGroup

    var sum1, sum2 int
    mid := len(arr) / 2

    wg.Add(2)
    go sumPart(arr[:mid], &sum1, &wg)
    go sumPart(arr[mid:], &sum2, &wg)

    // Čekamo da obe gorutine završe
    wg.Wait()
```

Konkurentno programiranje – primer

```
    // Konačna suma
    totalSum := sum1 + sum2
    fmt.Printf("Suma elemenata niza je: %d\n", totalSum)
}
```

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

- komponente mogu biti različitih vrsta, uključujući biblioteke, module ili čak mikroservise;

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

- komponente mogu biti različitih vrsta, uključujući biblioteke, module ili čak mikroservise;
- brži i efikasniji razvoj aplikacija;

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

- komponente mogu biti različitih vrsta, uključujući biblioteke, module ili čak mikroservise;
- brži i efikasniji razvoj aplikacija;
- slično objektno-orjentisanom programiranju, ali su komponente veće programske celine u odnosu na klase;

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

- komponente mogu biti različitih vrsta, uključujući biblioteke, module ili čak mikroservise;
- brži i efikasniji razvoj aplikacija;
- slično objektno-orjentisanom programiranju, ali su komponente veće programske celine u odnosu na klase;
- pristup *prevuci i postavi*

Komponentno programiranje

Komponentno programiranje

Komponentno programiranje je pristup razvoju softvera koji se fokusira na izgradnju aplikacija korišćenjem nezavisnih, ponovo upotrebljivih komponenti.

- komponente mogu biti različitih vrsta, uključujući biblioteke, module ili čak mikroservise;
- brži i efikasniji razvoj aplikacija;
- slično objektno-orjentisanom programiranju, ali su komponente veće programske celine u odnosu na klase;
- pristup *prevuci i postavi*
- može ostvariti u različitim programskim jezicima opšte namene (npr. C, C++, Java, C#, Swift), kroz razvojna okruženja i biblioteke.

Vizuelno programiranje

Vizuelno programiranje

Vizuelno programiranje koristi grafičko okruženje za kreiranje programa, umesto pisanja koda u tekstualnom obliku. U vizuelnom programiranju, programeri koriste vizuelne elemente kao što su blokovi, ikone, linije i dijagrami za pravljenje programa.

Vizuelno programiranje

Vizuelno programiranje

Vizuelno programiranje koristi grafičko okruženje za kreiranje programa, umesto pisanja koda u tekstualnom obliku. U vizuelnom programiranju, programeri koriste vizuelne elemente kao što su blokovi, ikone, linije i dijagrami za pravljenje programa.

- *Scratch* i *Blockly* – pogodno za početnike

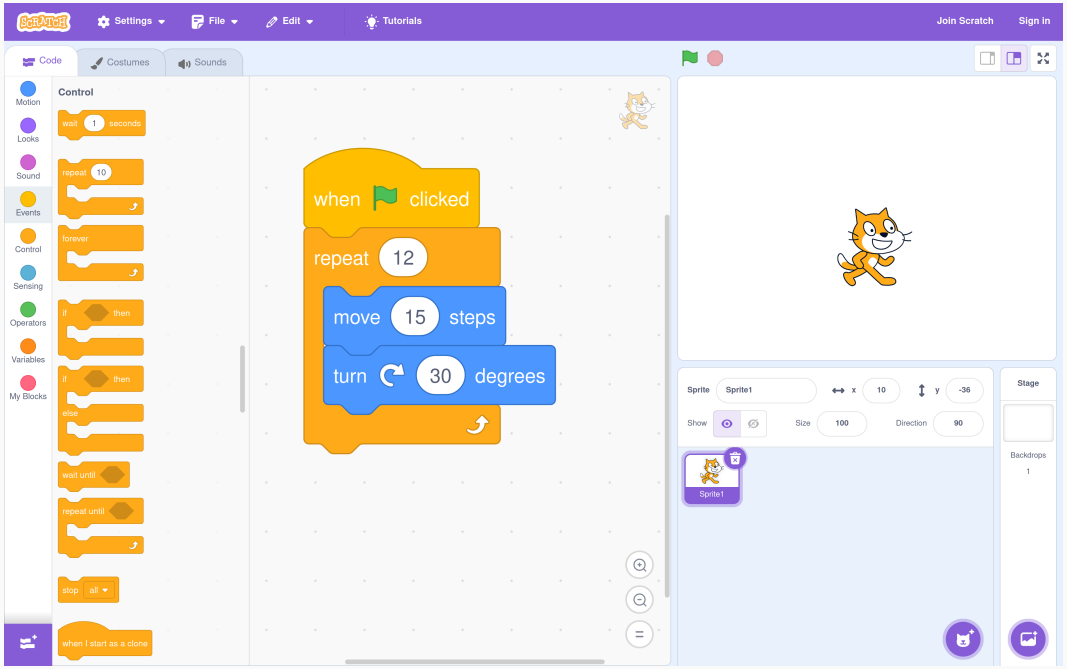
Vizuelno programiranje

Vizuelno programiranje

Vizuelno programiranje koristi grafičko okruženje za kreiranje programa, umesto pisanja koda u tekstualnom obliku. U vizuelnom programiranju, programeri koriste vizuelne elemente kao što su blokovi, ikone, linije i dijagrami za pravljenje programa.

- *Scratch* i *Blockly* – pogodno za početnike
- *Node-RED*, *LabVIEW* i *Unreal Engine Blueprints* – u industrijskom okruženju

Vizuelno programiranje – Scratch primer



Generičko programiranje

Generičko programiranje je paradigma koja omogućava pisanje koda koji može da se upotrebljava sa različitim tipovima podataka. Osnovna ideja je da se algoritmi i strukture podataka mogu definisati na uopšten način, tako da rade sa bilo kojim tipom podataka, dok god taj tip zadovoljava određene zahteve.

Generičko programiranje

Generičko programiranje je paradigma koja omogućava pisanje koda koji može da se upotrebljava sa različitim tipovima podataka. Osnovna ideja je da se algoritmi i strukture podataka mogu definisati na uopšten način, tako da rade sa bilo kojim tipom podataka, dok god taj tip zadovoljava određene zahteve.

- obezbeđuje fleksibilnost i ponovnu upotrebljivost napisanog koda;

Generičko programiranje

Generičko programiranje je paradigma koja omogućava pisanje koda koji može da se upotrebljava sa različitim tipovima podataka. Osnovna ideja je da se algoritmi i strukture podataka mogu definisati na uopšten način, tako da rade sa bilo kojim tipom podataka, dok god taj tip zadovoljava određene zahteve.

- obezbeđuje fleksibilnost i ponovnu upotrebljivost napisanog koda;
- programski jezici koji podržavaju generičko programiranje su C++, Java i C#.

Generičko programiranje – C++ primer

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

template <typename T>
T maksimum(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    cout << maksimum(110, 210) << endl;
    cout << maksimum(5.7, 5.3) << endl;
    cout << maksimum('g', 'e') << endl;
    return 0;
}
```

Upitni jezici

Pripadaju deklarativnoj paradigmi. To su specijalizovani programski jezici koji se koriste za interakciju sa bazama podataka. Oni omogućavaju korisnicima da efikasno pretražuju, manipulišu i upravljaju podacima u različitim vrstama baza podataka.

Upitni jezici

Upitni jezici

Pripadaju deklarativnoj paradigmi. To su specijalizovani programski jezici koji se koriste za interakciju sa bazama podataka. Oni omogućavaju korisnicima da efikasno pretražuju, manipulišu i upravljaju podacima u različitim vrstama baza podataka.

- najpoznatiji upitni jezik je SQL, koji se koristi za rad sa *relacionim* bazama podataka;

Značajni upitni jezici

- SQL – *Structured Query Language*, 1970-ih godina

Značajni upitni jezici

- SQL – *Structured Query Language*, 1970-ih godina
  - osnovni alat za rad sa relacionim bazama podataka;

Značajni upitni jezici

- SQL – *Structured Query Language*, 1970-ih godina
  - osnovni alat za rad sa relacionim bazama podataka;
  - omogućavajući definisanje, manipulaciju i kontrolu podataka na jednostavan i efikasan način;

Značajni upitni jezici

- SQL – *Structured Query Language*, 1970-ih godina
  - osnovni alat za rad sa relacionim bazama podataka;
  - omogućavajući definisanje, manipulaciju i kontrolu podataka na jednostavan i efikasan način;
  - koristi se za poslovnu analitiku, razvoj najrazličitijih aplikacija i rad sa velikim količinama podataka

Značajni upitni jezici

- SQL – *Structured Query Language*, 1970-ih godina
  - osnovni alat za rad sa relacionim bazama podataka;
  - omogućavajući definisanje, manipulaciju i kontrolu podataka na jednostavan i efikasan način;
  - koristi se za poslovnu analitiku, razvoj najrazličitijih aplikacija i rad sa velikim količinama podataka
-

Značajni upitni jezici

- SPARQL – jezik za postavljanje upita nad RDF podacima (osnova za rad sa semantičkim vebom i povezanim podacima).

Značajni upitni jezici

- SPARQL – jezik za postavljanje upita nad RDF podacima (osnova za rad sa semantičkim vebom i povezanim podacima).
- XQuery – dizajniran za pretraživanje i manipulaciju XML podacima; efikasno pretraživanje i izvođenje informacije iz XML dokumenata, kao i transformacija tih podatka.

Upitni jezici – primer

SQL upit iz baze podataka koja sadrzi tabelu `Osobe` izvlači sve informacije o svim osobama starijim od 24 godine.

```
SELECT * FROM Osobe WHERE Godine > 24;
```

Programiranje ograničenja

Programiranje ograničenja je deklarativna paradigma u okviru koje je posao programera da detaljno opiše uslove u sistemu za koji se traži rešenje. Koristi se u domenu optimizacija i rešavanja kombinatornih problema.

Programiranje ograničenja

Programiranje ograničenja je deklarativna paradigma u okviru koje je posao programera da detaljno opiše uslove u sistemu za koji se traži rešenje. Koristi se u domenu optimizacija i rešavanja kombinatornih problema.

Veliki broj programskih jezika opšte namene ima biblioteke koje pružaju podršku za programiranje u ovom stilu (jezici C, C++, Java, Python, C# i mnogi drugi).

Leksika, sintaksa, semantika i pragmatika programskih jezika

Leksika, sintaksa, semantika i pragmatika programskih jezika

Pitanjima ispravnosti programa bavi se sintaksa programskih jezika i njena podoblast leksika programskih jezika.

Leksika, sintaksa, semantika i pragmatika programskih jezika

Pitanjima ispravnosti programa bavi se sintaksa programskih jezika i njena podoblast leksika programskih jezika.

Leksika se bavi opisivanjem osnovnih gradivnih elemenata jezika

Leksika, sintaksa, semantika i pragmatika programskih jezika

Pitanjima ispravnosti programa bavi se sintaksa programskih jezika i njena podoblast leksika programskih jezika.

Leksika se bavi opisivanjem osnovnih gradivnih elemenata jezika

Sintaksa programskog jezika se bavi načinima za kombinovanje tih osnovnih elemenata u ispravne jezičke konstrukcije

Leksika, sintaksa, semantika i pragmatika programskih jezika

Pitanjima ispravnosti programa bavi se sintaksa programskih jezika i njena podoblast leksika programskih jezika.

Leksika se bavi opisivanjem osnovnih gradivnih elemenata jezika

Sintaksa programskog jezika se bavi načinima za kombinovanje tih osnovnih elemenata u ispravne jezičke konstrukcije

Semantika programskih jezika

Pitanjem značenja programa bavi se *semantika programskih jezika*.

Leksika, sintaksa, semantika i pragmatika programskih jezika

Leksika

Leksika

- Osnovni leksički elementi prirodnih jezika su reči, pri čemu se razlikuje nekoliko različitih vrsta reči (imenice, glagoli, pridevi, . . .) i reči imaju različite oblike (padeži, vremena, . . .).

Leksika

- Osnovni leksički elementi prirodnih jezika su reči, pri čemu se razlikuje nekoliko različitih vrsta reči (imenice, glagoli, pridevi, . . .) i reči imaju različite oblike (padeži, vremena, . . .).
- Zadatak leksičke analize prirodnog jezika je da identifikuje reči u rečenici i svrsta ih u odgovarajuće kategorije.

Leksika

- Osnovni leksički elementi prirodnih jezika su reči, pri čemu se razlikuje nekoliko različitih vrsta reči (imenice, glagoli, pridevi, . . .) i reči imaju različite oblike (padeži, vremena, . . .).
- Zadatak leksičke analize prirodnog jezika je da identifikuje reči u rečenici i svrsta ih u odgovarajuće kategorije.
- Slično važi i za programske jezike.

Leksika

- Osnovni leksički elementi prirodnih jezika su reči, pri čemu se razlikuje nekoliko različitih vrsta reči (imenice, glagoli, pridevi, . . .) i reči imaju različite oblike (padeži, vremena, . . .).
- Zadatak leksičke analize prirodnog jezika je da identifikuje reči u rečenici i svrsta ih u odgovarajuće kategorije.
- Slično važi i za programske jezike.

Leksika

- Osnovni leksički elementi prirodnih jezika su reči, pri čemu se razlikuje nekoliko različitih vrsta reči (imenice, glagoli, pridevi, . . .) i reči imaju različite oblike (padeži, vremena, . . .).
- Zadatak leksičke analize prirodnog jezika je da identifikuje reči u rečenici i svrsta ih u odgovarajuće kategorije.
- Slično važi i za programske jezike.

Leksikom programa obično se bavi deo programskog prevodioca koji se naziva *leksički analizator*.

Leksika – primer

```
if (a < 3)
        x1 = 3+4*a;
```

`if`	ključna reč
`(`	zagrada
`a`	identifikator
`<`	operator
`3`	celobrojni literal
`)`	zagrada
`x1`	identifikator
`=`	operator
`3`	celobrojni literal
`+`	operator
`4`	celobrojni literal
`*`	operator
`a`	identifikator
`;`	interpunkcija

Sintaksa programskog jezika

- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.

Sintaksa programskog jezika

- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.
- Slično je i sa programskim jezicima, gde se umesto ispravnih rečenica razmatraju ispravni programi.

Sintaksa programskog jezika

- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.
- Slično je i sa programskim jezicima, gde se umesto ispravnih rečenica razmatraju ispravni programi.
- Sintaksa definiše formalne relacije između elemenata jezika, time pružajući strukturne opise ispravnih niski jezika.

Sintaksa programskog jezika

- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.
- Slično je i sa programskim jezicima, gde se umesto ispravnih rečenica razmatraju ispravni programi.
- Sintaksa definiše formalne relacije između elemenata jezika, time pružajući strukturne opise ispravnih niski jezika.
- Sintaksa se bavi samo formom i strukturom jezika bez bilo kakvih razmatranja u vezi sa njihovim značenjem.

Sintaksa programskog jezika

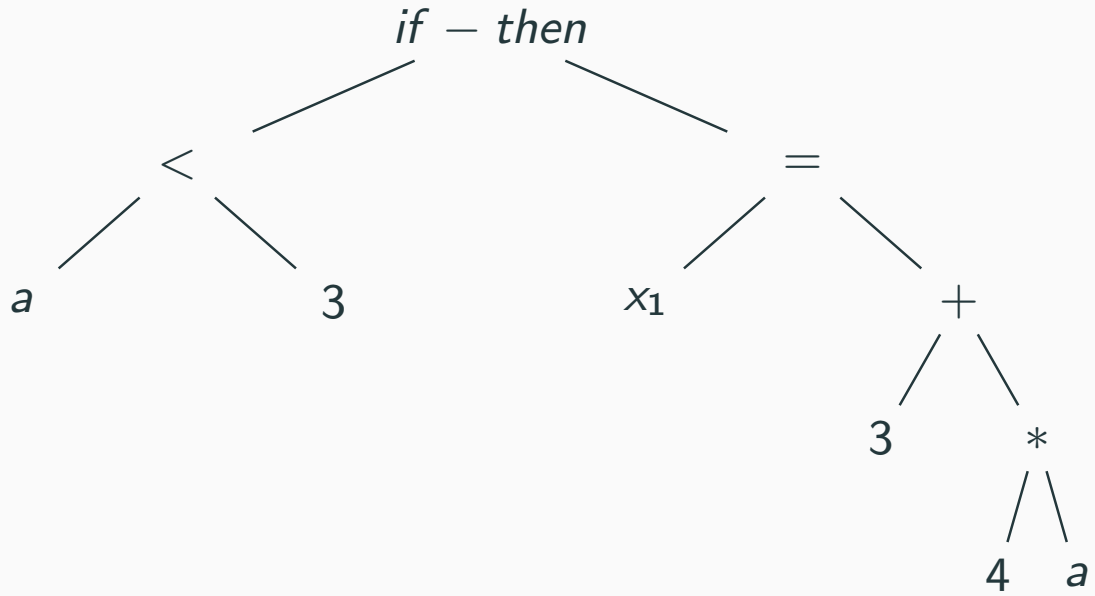
- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.
- Slično je i sa programskim jezicima, gde se umesto ispravnih rečenica razmatraju ispravni programi.
- Sintaksa definiše formalne relacije između elemenata jezika, time pružajući strukturne opise ispravnih niski jezika.
- Sintaksa se bavi samo formom i strukturom jezika bez bilo kakvih razmatranja u vezi sa njihovim značenjem.

Sintaksa programskog jezika

- Sintaksa prirodnih jezika definiše načine na koji pojedinačne reči mogu da kreiraju ispravne rečenice jezika.
- Slično je i sa programskim jezicima, gde se umesto ispravnih rečenica razmatraju ispravni programi.
- Sintaksa definiše formalne relacije između elemenata jezika, time pružajući strukturne opise ispravnih niski jezika.
- Sintaksa se bavi samo formom i strukturom jezika bez bilo kakvih razmatranja u vezi sa njihovim značenjem.

Sintaksička struktura rečenica ili programa se često predstavlja u obliku stabla.

Sintaksno stablo – primer



Sintaksom programa obično se bavi deo programskog prevodioca koji se naziva sintaksički analizator.

Leksika, sintaksa, semantika i pragmatika programskih jezika

Semantika

Semantika

Semantika pridružuje značenje sintaksički ispravnim niskama jezika.

Semantika za dati program opisuje koje je izračunavanje opisano tim programom.

Semantika – primer

```
int x = 0;
int x = 5;
```

Semantika – primer

```
int x = 0;
int x = 5;
```

Iako su oba reda sintaksno ispravna, kôd je semantički neispravan jer se dva puta deklariše promenjiva sa istim imenom.

Statička i dinamička semantika

Statička semantika

Neki aspekti semantičke korektnosti programa se mogu proveriti tokom prevođenja programa (na primer, da su sve promenljive koje se koriste u izrazima definisane i da su odgovarajućeg tipa).

Statička i dinamička semantika

Statička semantika

Neki aspekti semantičke korektnosti programa se mogu proveriti tokom prevođenja programa (na primer, da su sve promenljive koje se koriste u izrazima definisane i da su odgovarajućeg tipa).

Dinamička semantika

Neki aspekti mogu proveriti tek u fazi izvršavanja programa (na primer, da ne dolazi do deljenja nulom).

Semantika

Većina savremenih jezika ima precizno i formalno definisanu leksiku i sintaksu.

Semantika

Većina savremenih jezika ima precizno i formalno definisanu leksiku i sintaksu.

Formalna definicija semantike postoji samo za neke programske jezike (opisuje neformalno, opisima zadatim korišćenjem prirodnog jezika).

Semantika

Većina savremenih jezika ima precizno i formalno definisanu leksiku i sintaksu.

Formalna definicija semantike postoji samo za neke programske jezike (opisuje neformalno, opisima zadatim korišćenjem prirodnog jezika).

Semantika

Čest je slučaj da neki aspekti semantike ostaju nedefinisani standardom jezika i prepušta se implementacijama programskih prevodilaca da samostalno odrede potpunu semantiku.

Semantika

Čest je slučaj da neki aspekti semantike ostaju nedefinisani standardom jezika i prepušta se implementacijama programskih prevodilaca da samostalno odrede potpunu semantiku.

Primer u C-u: f() + g()

Nije definisano koja funkcija ce prva biti pozvana.

Leksika, sintaksa, semantika i pragmatika programskih jezika

Pragmatika programskih jezika

Pragmatika programskih jezika

Pragmatika jezika govori o izražajnosti jezika i o odnosu različitih načina za iskazivanje istih stvari.

Pragmatika programskih jezika

Pragmatika jezika govori o izražajnosti jezika i o odnosu različitih načina za iskazivanje istih stvari.

> Pragmatika programskih jezika uključuje pitanja kao što su lakoća programiranja, efikasnost u primenama i metodologija programiranja.

Pragmatika programskih jezika

Pragmatika jezika govori o izražajnosti jezika i o odnosu različitih načina za iskazivanje istih stvari.

> Pragmatika programskih jezika uključuje pitanja kao što su lakoća programiranja, efikasnost u primenama i metodologija programiranja.

U kontekstu programera, tj. kako se jezik koristi u praksi za postizanje specifičnih ciljeva, kao što su efikasnost, čitljivost i održivost koda.

Pragmatika se bavi i pitanjem kako i zašto programeri koriste određene elemente jezika u različitim kontekstima.

Pragmatika programskih jezika – primer1

```
for(i = 0; i < n; i++) {
    cout << i << " ";
}
```

```
while(broj != 0) {
    cin >> broj;
    suma += broj;
}
```

Pragmatika programskih jezika – primer2

```
if (broj % 2 == 0)
    cout << "Broj je paran!" << endl;
else
    cout << "Broj je neparan!" << endl;


switch (broj % 5) {
    case 1: cout << "Jedan" << endl; break;
    case 2: cout << "Dva" << endl; break;
    case 3: cout << "Tri" << endl; break;
    case 4: cout << "Cetiri" << endl; break;
    default: cout << "Broj je deljiv sa 5" << endl;
}
```

Pragmatika programskih jezika – primer3

Programeri koriste funkcije da bi smanjili dupliranje koda i olakšali održavanje. Funkcije omogućavaju lakšu organizaciju koda, ponovnu upotrebu i modularnost.

Pragmatika programskih jezika – primer4

```
int faktorijelRekurzivno(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * faktorijelRekurzivno(n - 1);
}


int faktorijelIterativno(int n) {
    int rezultat = 1;
    for (int i = 1; i <= n; i++) {
        rezultat *= i;
    }
    return rezultat;
}
```

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,
- načinima da se kreiraju kompleksni tipovi podataka,

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,
- načinima da se kreiraju kompleksni tipovi podataka,
- načini na koji se može ostvariti kontrola toka izvršavanja u programima,

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,
- načinima da se kreiraju kompleksni tipovi podataka,
- načini na koji se može ostvariti kontrola toka izvršavanja u programima,
- upotreba i karakteristike potprograma,

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,
- načinima da se kreiraju kompleksni tipovi podataka,
- načini na koji se može ostvariti kontrola toka izvršavanja u programima,
- upotreba i karakteristike potprograma,
- modularnost i načini omogućavanja podele koda i modularnog programiranja,

Pragmatika programskih jezika

Pragmatika programskih jezika može se razmatrati i u širem kontekstu:

> obuhvata najpre izbor jezika i tehnologije koji će se koristiti za rešavanje nekog konkretnog problema, a zatim i razumevanje koncepata dizajna programskih jezika.

Bavi se i pitanjima dizajna programskih jezika:

- izbor i karakteristike tipova podataka,
- načinima da se kreiraju kompleksni tipovi podataka,
- načini na koji se može ostvariti kontrola toka izvršavanja u programima,
- upotreba i karakteristike potprograma,
- modularnost i načini omogućavanja podele koda i modularnog programiranja,

Jezički procesori

Jezički procesori

Jezički procesori (ili programski prevodioci) su programi čija je uloga da analiziraju leksičku, sintaksičku i (donekle) semantičku ispravnost programa višeg programskog jezika i da na osnovu ispravnog ulaznog programa višeg programskog jezika generišu kôd na mašinskom jeziku.

Kompilatori

Ceo program analizira i transformiše u mašinski kôd pre nego što može da se izvrši.

Kompilatori

Ceo program analizira i transformiše u mašinski kôd pre nego što može da se izvrši.

Faza prevođenja i faza izvršavanja programa potpuno razdvojene.

Kompilatori

- Nakon analize izvornog koda programa višeg programskog jezika, kompilatori generišu izvršivi(mašinski) kôd i dodatno ga optimizuju, a zatim čuvaju u vidu izvršivih (binarnih) datoteka.

Kompilatori

- Nakon analize izvornog koda programa višeg programskog jezika, kompilatori generišu izvršivi(mašinski) kôd i dodatno ga optimizuju, a zatim čuvaju u vidu izvršivih (binarnih) datoteka.
- Jednom sačuvani mašinski kôd moguće je izvršavati neograničen broj puta.

Kompilatori

- Nakon analize izvornog koda programa višeg programskog jezika, kompilatori generišu izvršivi(mašinski) kôd i dodatno ga optimizuju, a zatim čuvaju u vidu izvršivih (binarnih) datoteka.
- Jednom sačuvani mašinski kôd moguće je izvršavati neograničen broj puta.
- Krajnjim korisnicima nije neophodno dostavljati izvorni kôd programa na višem programskom jeziku, već je dovoljno distribuirati izvršivi mašinski kôd.

Kompilatori

- Problem: prevođenjem se gubi svaka veza između izvornog i izvršivog koda programa.

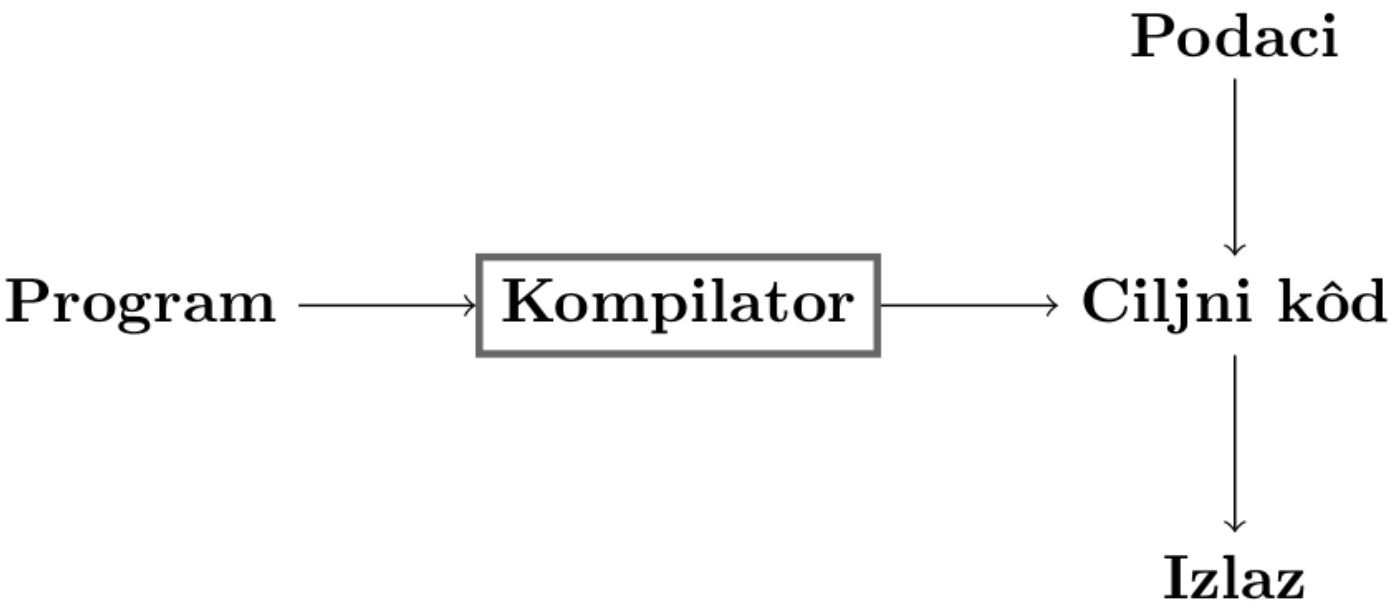
Kompilatori

- Problem: prevođenjem se gubi svaka veza između izvornog i izvršivog koda programa.
- Problem: Svaka (i najmanja) izmena u izvornom kodu programa zahteva ponovno prevođenje programa ili njegovih delova pri čemu samo prevođenje zahteva značajne, pre svega vremenske resurse.

Kompilatori

- Problem: prevođenjem se gubi svaka veza između izvornog i izvršivog koda programa.
- Problem: Svaka (i najmanja) izmena u izvornom kodu programa zahteva ponovno prevođenje programa ili njegovih delova pri čemu samo prevođenje zahteva značajne, pre svega vremenske resurse.
- Prednosti: kompilirani programi su obično veoma efikasni, značajno efikasniji nego kada se program izvršava ne neki od narednih načina.

Kompilatori



Slika 1.1: Kompilacija i izvršavanje

Interpretatori

Interpretatori su programski prevodioci kod kojih su faza prevođenja i faza izvršavanja programa isprepletane.

Interpretatori

Interpretatori su programski prevodioci kod kojih su faza prevođenja i faza izvršavanja programa isprepletane.

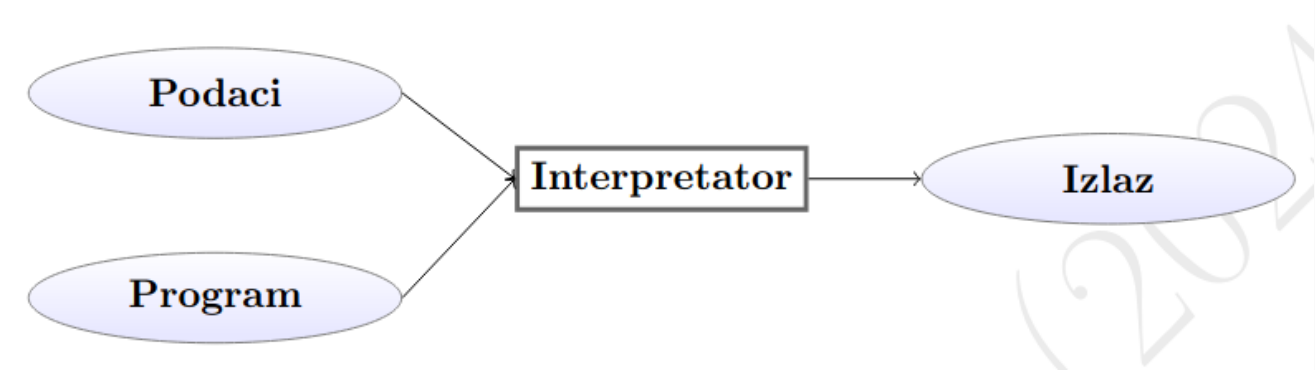
Interpretatori analiziraju deo po deo (najčešće naredbu po naredbu) izvornog koda programa i odmah nakon analize vrše i njegovo izvršavanje.

- Rezultat prevođenja se ne smešta u izvršive datoteke, već je prilikom svakog izvršavanja neophodno iznova vršiti analizu izvornog koda.

- Rezultat prevođenja se ne smešta u izvršive datoteke, već je prilikom svakog izvršavanja neophodno iznova vršiti analizu izvornog koda.
- Problem: programi koji se interpretiraju se obično izvršavaju znatno sporije nego u slučaju kompilacije.

- Rezultat prevođenja se ne smešta u izvršive datoteke, već je prilikom svakog izvršavanja neophodno iznova vršiti analizu izvornog koda.
- Problem: programi koji se interpretiraju se obično izvršavaju znatno sporije nego u slučaju kompilacije.
- Prednosti: razvojni ciklus programa je često kraći; prilikom malih izmena programa nije potrebno iznova vršiti analizu celokupnog koda

Interpretatori



Kombinacija kompilacije i interpretacije

- Danas se često primenjuje i tehnika kombinovanja kompilatora i interpretatora.

Kombinacija kompilacije i interpretacije

- Danas se često primenjuje i tehnika kombinovanja kompilatora i interpretatora.
- Kôd sa višeg programskog jezika se **prvo kompilira** u neki precizno definisan **međujezik** niskog nivoa, a zatim se vrši interpretacija ili kompilacija u vreme izvršavanja ovog međujezika i njegovo izvršavanje na konkretnom računaru.

Kombinacija kompilacije i interpretacije

- Neke platforme i programski jezici zasnovani su na ideji da se programi kompiliraju u specifičan „poluprevedeni kôd“ (bajtkod).

Kombinacija kompilacije i interpretacije

- Neke platforme i programski jezici zasnovani su na ideji da se programi kompiliraju u specifičan „poluprevedeni kôd“ (bajtkod).
- Zatim se taj kôd prilikom izvršavanja interpretira ili kompilira na neki specifičan način.

Bajtkod

Bajtkod

Bajtkod se može shvatiti kao asemblerski tj. mašinski jezik neke *virtuelne mašine* koji je mnogo nižeg nivoa nego originalni program na višem programskom jeziku, ali koji za razliku od realnog asemblera ne pokriva detalje konkretne arhitekture na kojoj će se program izvršavati.

Bajtkod

Bajtkod

Bajtkod se može shvatiti kao asemblerski tj. mašinski jezik neke *virtuelne mašine* koji je mnogo nižeg nivoa nego originalni program na višem programskom jeziku, ali koji za razliku od realnog asemblera ne pokriva detalje konkretne arhitekture na kojoj će se program izvršavati.

JIT kompilacija

JIT kompilacija

JIT kompilacija (engl. *just in time compilation*) – bajtkod se izvršava tako što se često izvršavane naredbe programa prevode u mašinske instrukcije za ciljnu mašinu, koje se onda čuvaju i direktno izvršavaju (umesto da se interpretiraju).

AOT kompilacija

AOT kompilacija

AOT kompilacija (engl. *ahead of time compilation*) koja podrazumeva da se pre svog izvršavanja ceo bajtkod prevede na mašinski jezik ciljnog računara.

Kombinacija kompilacije i interpretacije

- JIT i AOT kompilacijom se dobija mnogo brže izvršavanje programa nego kod klasičnog interpretiranja bajtkoda.

Kombinacija kompilacije i interpretacije

- JIT i AOT kompilacijom se dobija mnogo brže izvršavanje programa nego kod klasičnog interpretiranja bajtkoda.
- Prednost je jednostavna prenosivost programa prevedenih na bajtkod na različite platforme — za svaku novu platformu potrebno je izgraditi samo interpreter (mnogo jednostavnije nego razviti kompilator za polazni izvorni jezik visokog nivoa).

Kombinacija kompilacije i interpretacije

- Među najznačajnijim jezicima koji danas koriste bajtkod su Java i C#.

Kombinacija kompilacije i interpretacije

- Među najznačajnijim jezicima koji danas koriste bajtkod su Java i C#.
- Java programi mogu da se kompiliraju na takozvani JAVA bajtkod, koji se onda može interpretirati ili JIT kompilirati na bilo kakvom računaru (koji ima raspoloživu takozvanu Java virtualnu mašinu — JVM).

Kombinacija kompilacije i interpretacije

- Među najznačajnijim jezicima koji danas koriste bajtkod su Java i C#.
- Java programi mogu da se kompiliraju na takozvani JAVA bajtkod, koji se onda može interpretirati ili JIT kompilirati na bilo kakvom računaru (koji ima raspoloživu takozvanu Java virtualnu mašinu — JVM).
- U toku je i aktivni razvoj AOT kompilatora za Javu.

Kombinacija kompilacije i interpretacije

- Među najznačajnijim jezicima koji danas koriste bajtkod su Java i C#.
- Java programi mogu da se kompiliraju na takozvani JAVA bajtkod, koji se onda može interpretirati ili JIT kompilirati na bilo kakvom računaru (koji ima raspoloživu takozvanu Java virtualnu mašinu — JVM).
- U toku je i aktivni razvoj AOT kompilatora za Javu.
- Postoje prevodioci i za druge programske jezike koji koriste ovaj pristup i generišu Java bajtkod (na primer, programski jezici Scala i Kotlin).

Kombinacija kompilacije i interpretacije

- C# je deo Microsoft-ove .NET platforme i on se, kao i Visual Basic i F#, prevodi na bajtkod platforme .NET.

Kombinacija kompilacije i interpretacije

- C# je deo Microsoft-ove .NET platforme i on se, kao i Visual Basic i F#, prevodi na bajtkod platforme .NET.
- Dominantno vezana za operativni sistem Windows.

Jezički procesori

- Sâm jezik ne određuje da li će programi na njemu biti prevođeni na jedan, drugi ili neki treći način.

Jezički procesori

- Sâm jezik ne određuje da li će programi na njemu biti prevođeni na jedan, drugi ili neki treći način.
- Intepretator se često koristi u fazi razvoja programa da bi omogućio interakciju korisnika sa programom.

Jezički procesori

- Sâm jezik ne određuje da li će programi na njemu biti prevođeni na jedan, drugi ili neki treći način.
- Intepretator se često koristi u fazi razvoja programa da bi omogućio interakciju korisnika sa programom.
- U fazi eksploatacije kompletno razvijenog i istestiranog programa koristi kompilator koji proizvodi program sa efikasnim izvršavanjem.

Jezički procesori

Struktura kompilatora

Struktura kompilatora

- Prevođenje (kompilacija) programskog jezika je transformisanje teksta programa na jednom računarskom jeziku (viši programski jezik) u tekst programa na drugom jeziku (asembler, mašinski, bajtjezik).

Struktura kompilatora

- Prevođenje (kompilacija) programskog jezika je transformisanje teksta programa na jednom računarskom jeziku (viši programski jezik) u tekst programa na drugom jeziku (asembler, mašinski, bajtjezik).
- Kompilatori prevode čitav program na jeziku višeg nivoa u program na mašinskom ili nekom drugom ciljnom jeziku.

Struktura kompilatora

- Prevođenje (kompilacija) programskog jezika je transformisanje teksta programa na jednom računarskom jeziku (viši programski jezik) u tekst programa na drugom jeziku (asembler, mašinski, bajtjezik).
- Kompilatori prevode čitav program na jeziku višeg nivoa u program na mašinskom ili nekom drugom ciljnom jeziku.
- Ukoliko je ciljni jezik mašinski jezik, onda **kompilacija mora zavisiti od mašine na kojoj će se program izvršavati**.

Struktura kompilatora

- Prevođenje (kompilacija) programskog jezika je transformisanje teksta programa na jednom računarskom jeziku (viši programski jezik) u tekst programa na drugom jeziku (asembler, mašinski, bajtjezik).
- Kompilatori prevode čitav program na jeziku višeg nivoa u program na mašinskom ili nekom drugom ciljnom jeziku.
- Ukoliko je ciljni jezik mašinski jezik, onda **kompilacija mora zavisiti od mašine na kojoj će se program izvršavati**.
- Današnji kompilatori obično imaju tri ključne komponente.

Struktura kompilatora

Prednji sloj

Prednji sloj (eng. *front-end*) čita program zapisan na višem programskom jeziku, obrađuje ga i pohranjuje u obliku interne reprezentacije, međukoda.

Sastoji se od sledećih komponenti:

- Leksički analizator;
- Sintaksički analizator;
- Semantički analizator;
- Generator međukoda.

Srednji sloj

Srednji sloj (eng. *middle-end*) optimizije međukod i priprema ga za prevođenje na ciljni jezik.

Čini ga jedna komponenta:

- Optimizator međukoda.

Srednji sloj

Srednji sloj (eng. *middle-end*) optimizije međukod i priprema ga za prevođenje na ciljni jezik.

Čini ga jedna komponenta:

- Optimizator međukoda.

Iako se srednji sloj bavi samo jednim poslom, tj. optimizacijom međukoda, ovaj sloj je najkompleksniji i najvažniji sloj modernih kompilatora jer od njega najviše zavisi efikasnost izvršivog koda.

Struktura kompilatora

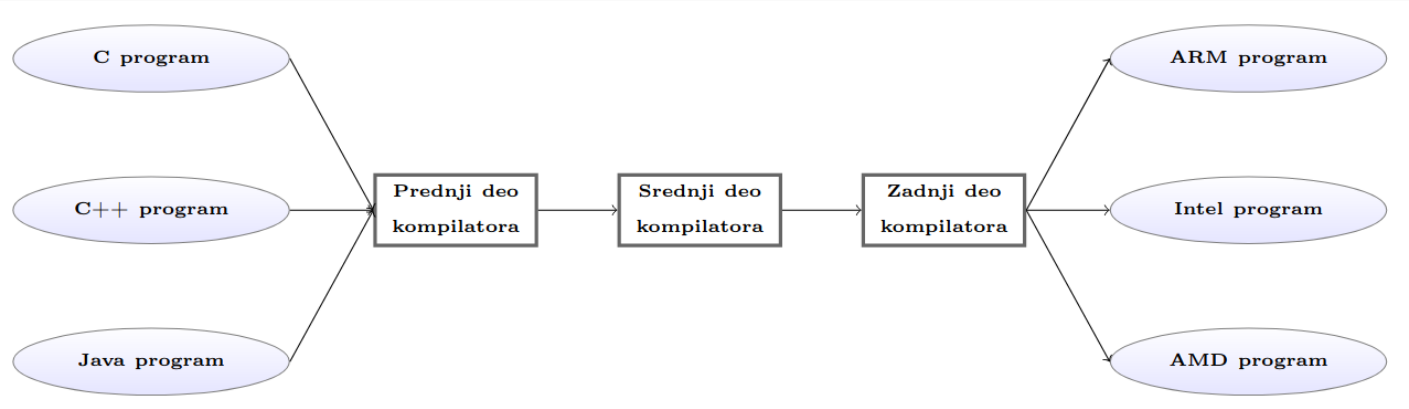
Zadnji sloj

Zadnji sloj] (eng. *back-end*) prevodi internu reprezentaciju (međukod) u ciljni jezik.

Čini ga naredna komponenta:

- Generator i optimizator ciljnog koda.

Struktura kompilatora



Jezički procesori

Leksička analiza

Leksička analiza

Leksički analizator

Leksička analiza je proces izdvajanja *tokena*, osnovnih jezičkih elemenata, iz niza ulaznih karaktera (na primer, karaktera koji čine program). Deo kompilacije koji se bavi leksičkom analizom naziva se **leksički analizator** ili **lekser**.

Leksička analiza

Leksički analizator

Leksička analiza je proces izdvajanja *tokena*, osnovnih jezičkih elemenata, iz niza ulaznih karaktera (na primer, karaktera koji čine program). Deo kompilacije koji se bavi leksičkom analizom naziva se **leksički analizator** ili **lekser**.

Token je sintaksička klasa, kategorija. Leksema je konkretan primerak, konkretna instanca jednog tokena.

Leksička analiza – primer

```
  if (starost >= 18)
      dopuna = 0;
  else
      dopuna = 18 - starost;
```

```
if (starost >= 18)\n\tdopnua=0;\nelse\n\tdopuna = 18 - staro
```

Leksički analizator

- Leksički analizator može imati i druge zadatke, kao što je, na primer, eliminacija komentara (ako nema pretprocesora).

Leksički analizator

- Leksički analizator može imati i druge zadatke, kao što je, na primer, eliminacija komentara (ako nema pretprocesora).
- Tokom leksičke analize u specijalnu tabelu koja se naziva tabela simbola, upisuju se prepoznati identifikatori i pridružuju im se određene relevantne informacije.

Leksički analizator

- Leksički analizator može imati i druge zadatke, kao što je, na primer, eliminacija komentara (ako nema pretprocesora).
- Tokom leksičke analize u specijalnu tabelu koja se naziva tabela simbola, upisuju se prepoznati identifikatori i pridružuju im se određene relevantne informacije.
- Ova tabela dopunjuje se tokom narednih faza kompilacije (na primer, informacijama o tipovima).

Leksički analizator

Leksička analiza može da otkrije neke (jednostavne) vrste grešaka u kodu.

```
int a = 09;  /* error: invalid digit "9" in octal constant
printf("Hi); /* error: missing terminating " character */
```

Regularni izrazi

karakterske klase: navode se između `[` i `]` i označavaju jedan od navedenih karaktera.

alternacija: navodi se sa `|` i označava alternativne mogućnosti.

opciono pojavljivanje: navodi se sa `?`.

ponavljanje: navodi se sa `*` i označava da se nešto javlja nula ili više puta.

pozitivno ponavljanje: navodi se sa `+` i označava da se nešto javlja jedan ili više puta.

Regularni izrazi – primeri

- primer karakterske klase: `[0-9]` označava cifru.

Regularni izrazi – primeri

- primer karakterske klase: `[0-9]` označava cifru.
- primer alternacije: `a|b` označava slovo `a` ili slovo `b`;

Regularni izrazi – primeri

- primer karakterske klase: `[0-9]` označava cifru.
- primer alternacije: `a|b` označava slovo `a` ili slovo `b`;
- primer opcionog pojavljivanja: `a?` označava da slovo `a` može, a ne mora da se javi;

Regularni izrazi – primeri

- primer karakterske klase: `[0-9]` označava cifru.
- primer alternacije: `a|b` označava slovo `a` ili slovo `b`;
- primer opcionog pojavljivanja: `a?` označava da slovo `a` može, a ne mora da se javi;
- prime ponavljanja: `a*` označava niz od nula ili više slova `a`;

Regularni izrazi – primeri

- primer karakterske klase: `[0-9]` označava cifru.
- primer alternacije: `a|b` označava slovo `a` ili slovo `b`;
- primer opcionog pojavljivanja: `a?` označava da slovo `a` može, a ne mora da se javi;
- prime ponavljanja: `a*` označava niz od nula ili više slova `a`;
- primer pozitivnog ponavljanja: `[0-9]+` označava neprazan niz cifara.

Regularni izrazi – primeri

Identifikatori u C++: neprazne niske koje se sastoje od slova, cifara i podvlaka, pri čemu ne mogu da počnu cifrom.

Regularni izrazi – primeri

Identifikatori u C++: neprazne niske koje se sastoje od slova, cifara i podvlaka, pri čemu ne mogu da počnu cifrom.

```
([a-zA-Z]|_)([a-zA-Z]|[0-9]|_)*
```

Regularni izrazi – primeri

Identifikatori u C++: neprazne niske koje se sastoje od slova, cifara i podvlaka, pri čemu ne mogu da počnu cifrom.

```
([a-zA-Z]|_)([a-zA-Z]|[0-9]|_)*
```

```
[a-zA-Z_][a-zA-Z_0-9]*
```

Regularni izrazi

- Algoritam za izdvajanje leksema iz ulaznog teksta zasniva se na konačnim automatima.

Regularni izrazi

- Algoritam za izdvajanje leksema iz ulaznog teksta zasniva se na konačnim automatima.
- Postoje programi koji na osnovu opisa tokena u vidu regularnih izraza generišu leksere na izabranom programskom jeziku. Primer takvog programa je `lex`.

Regularni izrazi

- Algoritam za izdvajanje leksema iz ulaznog teksta zasniva se na konačnim automatima.
- Postoje programi koji na osnovu opisa tokena u vidu regularnih izraza generišu leksere na izabranom programskom jeziku. Primer takvog programa je `lex`.
- Regularni izrazi ipak ne mogu da opišu baš sve — na primer, nije moguće napisati regularni izraz kojim bi se opisali svi ispravni aritmetički izrazi, tj. skup $\{a,\ a+a,\ a*a,\ a+a*a,\ a*(a+a),\ \ldots\}$.

Jezički procesori

Sintaksička analiza

Sintaksička analiza

Sintaksička analiza, poznata i kao *parsiranje*, je proces organizovanja leksema izdvojenih u fazi leksičke analize u ispravnu jezičku konstrukciju.

Sintaksička analiza

Sintaksička analiza, poznata i kao *parsiranje*, je proces organizovanja leksema izdvojenih u fazi leksičke analize u ispravnu jezičku konstrukciju.

Programi koji vrše parsiranje zovu se sintaksički analizatori ili parseri.

Sintaksička analiza

Sintaksička analiza, poznata i kao *parsiranje*, je proces organizovanja leksema izdvojenih u fazi leksičke analize u ispravnu jezičku konstrukciju.

Programi koji vrše parsiranje zovu se sintaksički analizatori ili parseri.

Rezultat sintaksičke analize za ispravnu ulaznu jezičku konstrukciju je sintaksičko stablo (ili stablo parsiranja).

Sintaksička analiza

Sintaksička analiza može da otkrije raznovrsne greške u kodu.

```
18 - starost = dopuna; /* error: lvalue required as left
                          operand of assignment */
if x x++;    /* error: expected ‘(’ before ‘x’ */
```

Načini opisa sintakse programskih jezika

Za opisivanje sintakse jezika obično se koriste kontekstno-slobodne gramatike:

- BNF (Bakus-Naurova forma)

Načini opisa sintakse programskih jezika

Za opisivanje sintakse jezika obično se koriste kontekstno-slobodne gramatike:

- BNF (Bakus-Naurova forma)
- EBNF (proširena Bakus-Naurova forma) (proširuje BNF operacijama regularnih izraza)

Načini opisa sintakse programskih jezika

Za opisivanje sintakse jezika obično se koriste kontekstno-slobodne gramatike:

- BNF (Bakus-Naurova forma)
- EBNF (proširena Bakus-Naurova forma) (proširuje BNF operacijama regularnih izraza)
- sintaksički dijagrami

Načini opisa sintakse programskih jezika

- Sintaksička analiza na osnovu zadate sintakse jezika zasniva se na takozvanim potisnim automatima.

Načini opisa sintakse programskih jezika

- Sintaksička analiza na osnovu zadate sintakse jezika zasniva se na takozvanim potisnim automatima.
- Postoje programi koji na osnovu opisa sintakse jezika (na primer, u vidu EBNF-a) generišu parsere na izabranom programskom jeziku.
  Primer takvog programa je program `Yacc` („yet another compiler compiler“).

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su izražajniji formalizam od regularnih izraza.

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su izražajniji formalizam od regularnih izraza.
- Sve što je moguće opisati regularnim izrazima, moguće je opisati i kontekstno-slobodnim gramatikama, (doduše regularni izrazi obično daju koncizniji opis).

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su određene skupom pravila.

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su određene skupom pravila.
- Sa leve strane pravila nalaze se takozvani pomoćni simboli (neterminali), dok se sa desne strane nalaze niske u kojima mogu da se javljaju bilo pomoćni simboli bilo takozvani završni simboli(terminali).

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su određene skupom pravila.
- Sa leve strane pravila nalaze se takozvani pomoćni simboli (neterminali), dok se sa desne strane nalaze niske u kojima mogu da se javljaju bilo pomoćni simboli bilo takozvani završni simboli(terminali).
- Jedan od pomoćnih simbola se smatra istaknutim, naziva se početnim simbolom (ili aksiomom).

Kontekstno-slobodne gramatike

- Kontekstno-slobodne gramatike su određene skupom pravila.
- Sa leve strane pravila nalaze se takozvani pomoćni simboli (neterminali), dok se sa desne strane nalaze niske u kojima mogu da se javljaju bilo pomoćni simboli bilo takozvani završni simboli(terminali).
- Jedan od pomoćnih simbola se smatra istaknutim, naziva se početnim simbolom (ili aksiomom).
- Niska je opisana gramatikom ako ju je moguće dobiti krenuvši od početnog simbola, zamenjujući u svakom koraku pomoćne simbole desnim stranama pravila.

Kontekstno-slobodne gramatike – primer 1

Identifikatora programskog jezika C:

$$
\begin{aligned}
I &\to XZ \\
X &\to S \mid P \\
Z &\to YZ \mid \varepsilon \\
Y &\to S \mid P \mid C \\
S &\to \texttt{a} \mid \ldots \mid \texttt{z} \mid \texttt{A} \mid \ldots \mid \texttt{Z} \\
P &\to \_ \\
C &\to 0 \mid \ldots \mid 9
\end{aligned}
$$

Kontekstno-slobodne gramatike – primer 1

Na primer, identifikator `x_1` moguće je izvesti na sledeći način:

$$I \Rightarrow XZ \Rightarrow SZ \Rightarrow \mathtt{x}Z \Rightarrow \mathtt{x}YZ \Rightarrow \mathtt{x}PZ \Rightarrow \mathtt{x\_}Z \Rightarrow$$

$$\mathtt{x\_}YZ \Rightarrow \mathtt{x\_}CZ \Rightarrow \mathtt{x\_1}Z \Rightarrow \mathtt{x\_1}.$$

Kontekstno-slobodne gramatike – primer 2

Ispravni aritmetički izrazi u kojima su dopuštene operacije sabiranja i množenja:

$$E \rightarrow E + E \mid E * E \mid (E) \mid a\,.$$

Kontekstno-slobodne gramatike – primer 2

Ispravni aritmetički izrazi u kojima su dopuštene operacije sabiranja i množenja:

$$E \rightarrow E + E \mid E * E \mid (E) \mid a\,.$$

Kontekstno-slobodne gramatike – primer 2

Izraz $a + a * a$ može se izvesti:

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a.$$

ili

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a.$$

Kontekstno-slobodne gramatike – primer 3

Ispravni aritmetički izrazi:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$

Preciznija – jednoznačno određen prioritet i asocijativnost operatora.

Kontekstno-slobodne gramatike – primer 3

Izraz $a + a * a$ može se izvesti:

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F \Rightarrow$$

$$a + F * F \Rightarrow a + a * F \Rightarrow a + a * a.$$

BNF – primer 1

Celi brojevi u dekadnom brojnom sistemu:

```
<ceo broj>             ::= <neoznacen ceo broj> |
                           <znak broja><neoznacen ceo broj>
<neoznacen ceo broj>   ::= <cifra> |
                           <neoznacen ceo broj><cifra>
<cifra>                ::= 0|1|2|3|4|5|6|7|8|9
<znak broja>           ::= +|-
```

BNF – primer 2

Aritmetičkih izrazi:

```
<izraz>  ::= <izraz> + <term> | <term>
<term>   ::= <term> * <faktor> | <faktor>
<faktor> ::= ( <izraz> ) | <ceo broj>
```

EBNF – primer 1

Identifikator u C-u:

```
<identifikator> ::= <slovo ili _> { <slovo ili _> |
                                    <cifra> }
<slovo ili _>   ::= "a"|...|"z"|"A"|...|"Z"|"_"
<cifra>         ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

EBNF – primer 2

Celi brojevi u dekadnom brojnom sistemu:

```
<ceo broj> ::= ["+"|"-"]<cifra>{<cifra>}
<cifra>    ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

EBNF – primer 3

Aritmetičkih izrazi:

```
<izraz>  ::= <term> {"+" <term>}
<term>   ::= <faktor> {"*" <faktor>}
<faktor> ::= "(" <izraz> ")" | <ceo broj>
```

EBNF – primer 4

Naredba grananja `if` sa opcionim pojavljivanjem `else`:

```
<if_naredba>  ::= if "(" <bulovski_izraz> ")"
                    <niz_naredbi>
                  [ else
                    <niz_naredbi> ]
<niz_naredbi> ::= "{" <naredba> ";" { <naredba> ";" } "}"
```

Jezički procesori

Semantička analiza

Semantička analiza

Semantička analiza je proces u kojem se proveravaju semantički uslovi i primenjuju pravila koja nije pogodno opisati sintaksičkim pravilima ni primenjivati u fazi sintaksičke analize.

Semantička analiza

Semantička analiza je proces u kojem se proveravaju semantički uslovi i primenjuju pravila koja nije pogodno opisati sintaksičkim pravilima ni primenjivati u fazi sintaksičke analize.

Semantička analiza vrši se nad sintaksičkim stablom.

Semantička analiza

Semantička analiza je proces u kojem se proveravaju semantički uslovi i primenjuju pravila koja nije pogodno opisati sintaksičkim pravilima ni primenjivati u fazi sintaksičke analize.

Semantička analiza vrši se nad sintaksičkim stablom.

Semantička analiza obično uključuje proveru tipova, implicitne konverzije, kao i provere koje se odnose na vidljivost promenljivih.

Semantička analiza

Semantička analiza može da otkrije raznovrsne greške u kodu:

```
int x, x;              /* error: redeclaration of ‘x’ with
                       no linkage */
char *s = x * "s"; /* error: invalid operands to binary
                          (have ‘int’ and ‘char *’) */
```

Jezički procesori

Generisanje međukoda

Generisanje međukoda

Generisanje međukoda je proces u okviru kojeg se izlaz iz faze semantičke analize (u vidu označenog sintaksičkog stabla) prevodi u linearnu reprezentaciju, nezavisnu od konkretnih mašina.

Generisanje međukoda

Generisanje međukoda je proces u okviru kojeg se izlaz iz faze semantičke analize (u vidu označenog sintaksičkog stabla) prevodi u linearnu reprezentaciju, nezavisnu od konkretnih mašina.

- Složeni izrazi mogu se svoditi u međukodu samo na pojedinačne operacije sa po dva argumenta, koje lako mogu da se prevedu na asembler.

Generisanje međukoda

Generisanje međukoda je proces u okviru kojeg se izlaz iz faze semantičke analize (u vidu označenog sintaksičkog stabla) prevodi u linearnu reprezentaciju, nezavisnu od konkretnih mašina.

- Složeni izrazi mogu se svoditi u međukodu samo na pojedinačne operacije sa po dva argumenta, koje lako mogu da se prevedu na asembler.
- Time je omogućeno da se (mnogi) izrazi u međukodu mogu čuvati kao nizovi jednostavnih četvorki koje čine operator, dva argumenta i rezultat.

Jezički procesori

Optimizacija međukoda

Optimizacija međukoda

Optimizacija međukoda je proces u okviru kojeg se na generisani međukod primenjuju raznovrsne optimizacije u cilju dobijanja efikasnijeg i kvalitetnijeg ciljnog koda, a bez promene njegovog vidljivog ponašanja (tj. uz čuvanje *semantike programa*).

Optimizacija međukoda

Optimizacija međukoda je proces u okviru kojeg se na generisani međukod primenjuju raznovrsne optimizacije u cilju dobijanja efikasnijeg i kvalitetnijeg ciljnog koda, a bez promene njegovog vidljivog ponašanja (tj. uz čuvanje *semantike programa*).

- Eliminisanje koda koji se ne koristi.

Optimizacija međukoda

Optimizacija međukoda je proces u okviru kojeg se na generisani međukod primenjuju raznovrsne optimizacije u cilju dobijanja efikasnijeg i kvalitetnijeg ciljnog koda, a bez promene njegovog vidljivog ponašanja (tj. uz čuvanje *semantike programa*).

- Eliminisanje koda koji se ne koristi.
- Propagacija konstanti.

Optimizacija međukoda

Optimizacija međukoda je proces u okviru kojeg se na generisani međukod primenjuju raznovrsne optimizacije u cilju dobijanja efikasnijeg i kvalitetnijeg ciljnog koda, a bez promene njegovog vidljivog ponašanja (tj. uz čuvanje *semantike programa*).

- Eliminisanje koda koji se ne koristi.
- Propagacija konstanti.
- Promena poretka naredbi.

Optimizacija međukoda

Optimizacija međukoda je proces u okviru kojeg se na generisani međukod primenjuju raznovrsne optimizacije u cilju dobijanja efikasnijeg i kvalitetnijeg ciljnog koda, a bez promene njegovog vidljivog ponašanja (tj. uz čuvanje *semantike programa*).

- Eliminisanje koda koji se ne koristi.
- Propagacija konstanti.
- Promena poretka naredbi.
- Transformacija petlji...

Optimizacija međukoda

- Optimizacije mogu da se odnose na procenjeno vreme izvšavanja, na procenjen prostor potreban za izvršavanje ili na procenjenu veličinu izvršivog programa.

Optimizacija međukoda

- Optimizacije mogu da se odnose na procenjeno vreme izvšavanja, na procenjen prostor potreban za izvršavanje ili na procenjenu veličinu izvršivog programa.
- Proces optimizacije mora da uzima u obzir procenjeno ukupno vreme kompilacije jer se često ne isplati primenjivati najkompleksnije optimizacije.

Optimizacija međukoda

- Optimizacije mogu da se odnose na procenjeno vreme izvšavanja, na procenjen prostor potreban za izvršavanje ili na procenjenu veličinu izvršivog programa.
- Proces optimizacije mora da uzima u obzir procenjeno ukupno vreme kompilacije jer se često ne isplati primenjivati najkompleksnije optimizacije.
- Mogu biti lokalne (najjednostavnije), da se odnose na razgranate delove programa (komplikovanije) ili one koje ispituju razlicite zavisnosti među funkcijama.

Optimizacija međukoda

- Optimizacije mogu da se odnose na procenjeno vreme izvšavanja, na procenjen prostor potreban za izvršavanje ili na procenjenu veličinu izvršivog programa.
- Proces optimizacije mora da uzima u obzir procenjeno ukupno vreme kompilacije jer se često ne isplati primenjivati najkompleksnije optimizacije.
- Mogu biti lokalne (najjednostavnije), da se odnose na razgranate delove programa (komplikovanije) ili one koje ispituju razlicite zavisnosti među funkcijama.
- U oblasti prevođenja programskih jezika, ovo je faza koja je najčešći predmet inovacija i istraživanja (prethodne faze se već dugo sprovode na suštinski iste načine).

Optimizacija međukoda – primer

```
x := x + 0  // eliminisanje koda bez efekta
x := x * 1  // eliminisanje koda bez efekta
x := x * 0  // moze se uprostiti
z := x + u  // moguca je primena propagacije vrednosti
y := 2 * x  // eliminisanje mrtvog koda
y := 2 * z  // moze se uprostiti
```

optimizovano:

```
x := 0
z := 0 + u;
y := z << 1
```

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,
- veličine izvršivog programa,

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,
- veličine izvršivog programa,
- energije koju program troši prilikom izvršavanja

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,
- veličine izvršivog programa,
- energije koju program troši prilikom izvršavanja

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,
- veličine izvršivog programa,
- energije koju program troši prilikom izvršavanja

Međutim, često su ovi ciljevi međusobno suprotstavljeni i to tako što ostvarenje jednog cilja negativno utiče na bar jedan od druga dva cilja.

Optimizacija međukoda – ciljevi

Tri osnovna cilja su smanjenje:

- vremena izvšavanja,
- veličine izvršivog programa,
- energije koju program troši prilikom izvršavanja

Međutim, često su ovi ciljevi međusobno suprotstavljeni i to tako što ostvarenje jednog cilja negativno utiče na bar jedan od druga dva cilja.

U oblasti prevođenja programskih jezika, faza optimizacije međukoda je najčešći predmet inovacija i istraživanja.

Jezički procesori

Generisanje i optimizacija ciljnog koda

Generisanje i optimizacija ciljnog koda

Generisanje i optimizacija ciljnog koda je proces prevođenja međukoda na ciljni jezik, često jezik prilagođen nekoj konkretnoj računarskoj arhitekturi.

Generisanje i optimizacija ciljnog koda

Generisanje i optimizacija ciljnog koda je proces prevođenja međukoda na ciljni jezik, često jezik prilagođen nekoj konkretnoj računarskoj arhitekturi.

Na primer:

- da li će neka promenljiva da bude čuvana u registrima ili u memoriji

Generisanje i optimizacija ciljnog koda

Generisanje i optimizacija ciljnog koda je proces prevođenja međukoda na ciljni jezik, često jezik prilagođen nekoj konkretnoj računarskoj arhitekturi.

Na primer:

- da li će neka promenljiva da bude čuvana u registrima ili u memoriji
- poredak instrukcija može biti promenjen radi kvalitetnijeg korišćenja procesora i registara.

Jezički procesori

Ilustracija sprovođenja faza kompilacije

Ilustracija sprovođenja faza kompilacije

```
cur_time = start_time + cycles * 60
```

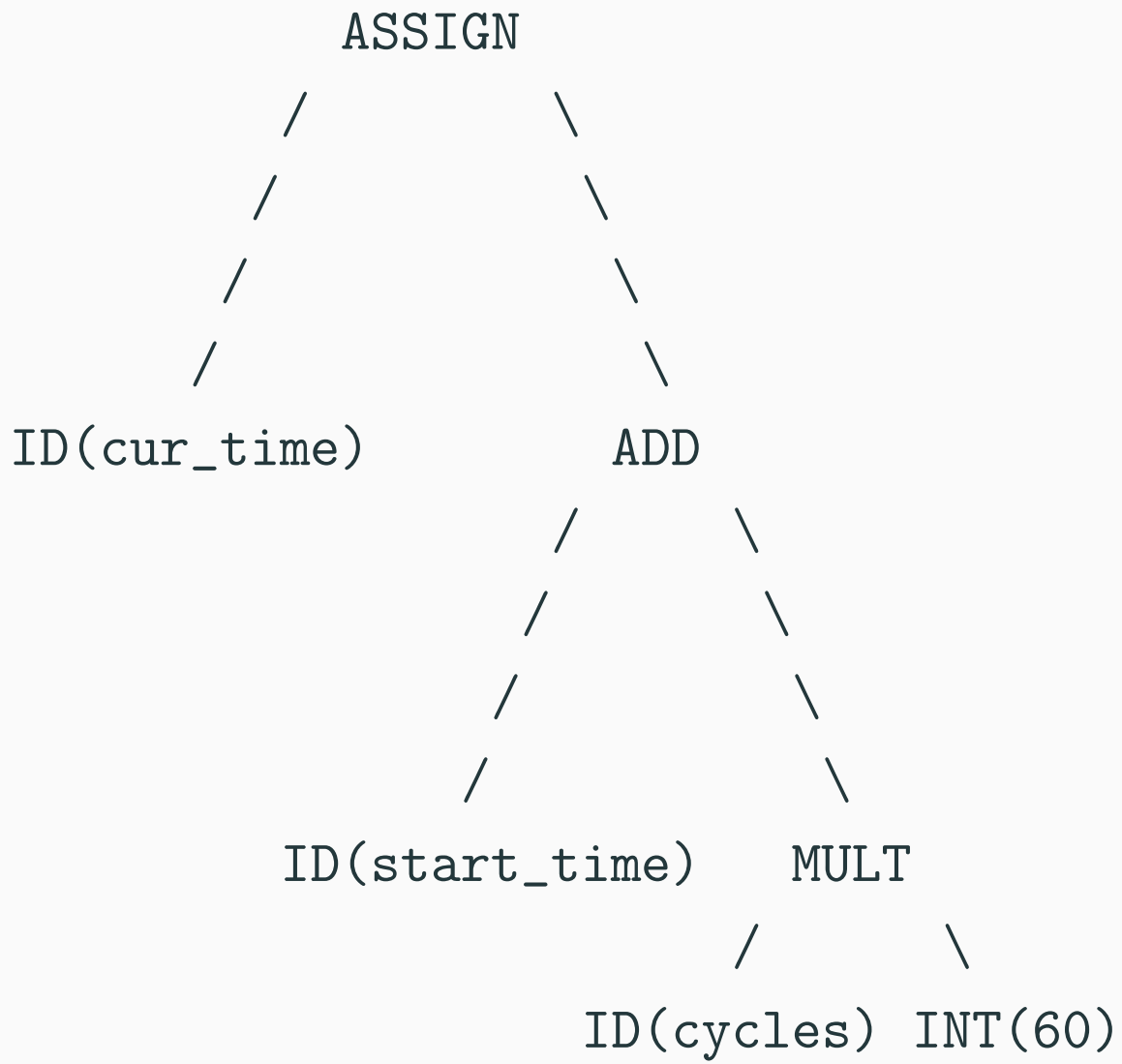
Ilustracija sprovođenja faza kompilacije

Leksička analiza: izdvajanje reči i njihovih kategorija (tri identifikatora, dodela, sabiranje, množenje i celobrojna konstanta)

```
ID(cur_time) ASSIGN ID(start_time) ADD ID(cycles) MULT
INT(60)
```

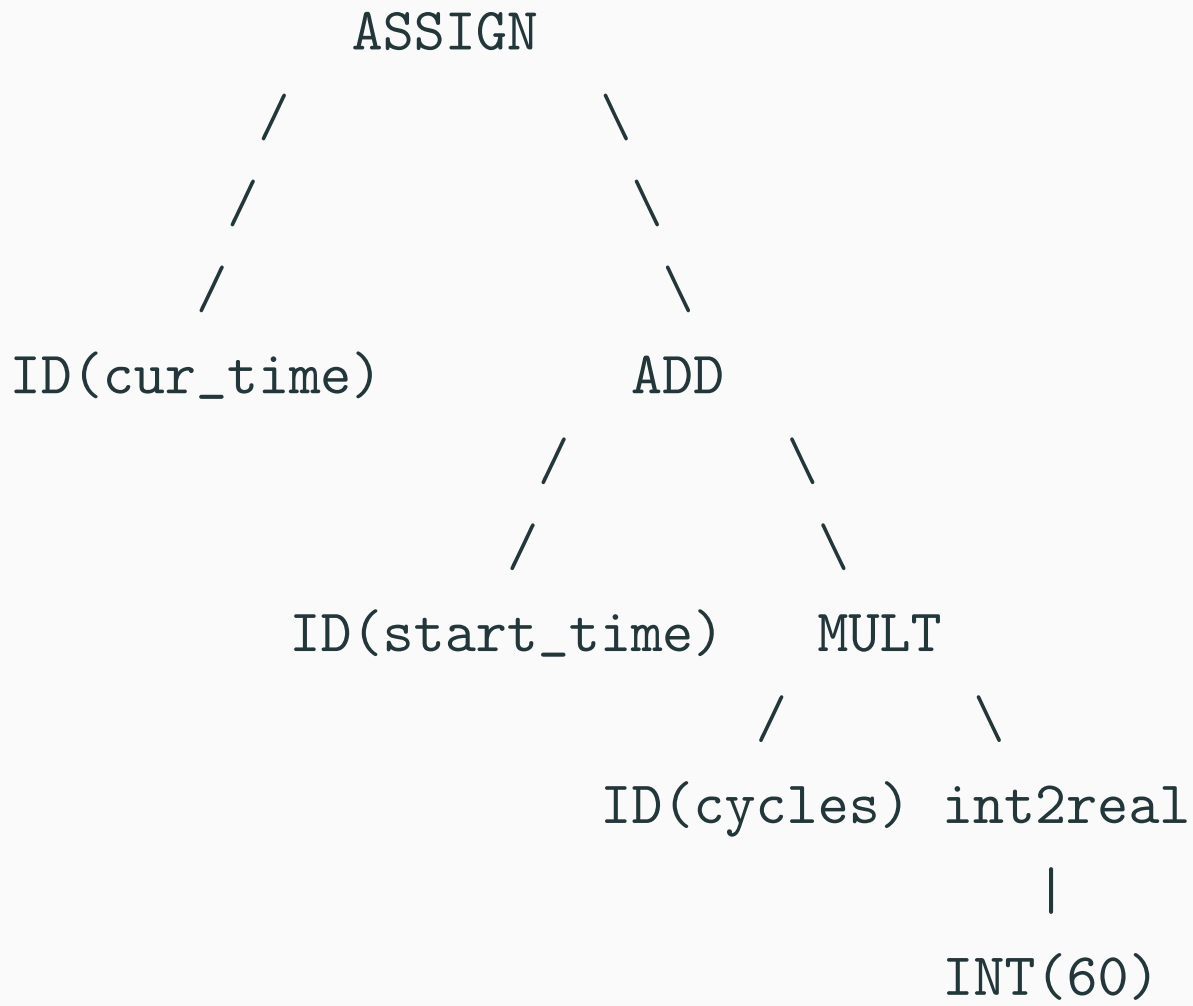
Ilustracija sprovođenja faza kompilacije

Sintaksička analiza: izgradnja sintaksičkog stabla



Ilustracija sprovođenja faza kompilacije

Semantička analiza: dodavanje čvora konverizije u sinstaksno stablo, usled nepoklapanja tipova



Ilustracija sprovođenja faza kompilacije

Generisanje međukoda: uvođenje privremenih promenljivih kako bi svaka naredba dodele imala najviše dva argumenta

```
tmp1 = int2real(60)
tmp2 = cycles * tmp1
tmp3 = start_time + tmp2
cur_time = tmp3
```

Ilustracija sprovođenja faza kompilacije

Optimizacija međukoda: zamena konverzije celobrojne promenljive 60 sa realnom promenljivom 60.0

```
tmp1 = 60.0
tmp2 = cycles * tmp1
tmp3 = start_time + tmp2
cur_time = tmp3
```

Ilustracija sprovođenja faza kompilacije

Optimizacija međukoda: zamena *tmp1* i *tmp3*

```
tmp1 = 60.0
tmp2 = cycles * 60.0
tmp3 = start_time + tmp2
cur_time = start_time + tmp2
```

Ilustracija sprovođenja faza kompilacije

Optimizacija međukoda: eliminacija mrtvog koda

```
tmp2 = cycles * 60.0
cur_time = start_time + tmp2
```

Ilustracija sprovođenja faza kompilacije

Generisanje koda na ciljnom jeziku:

```
MOVF R2, cycles       # Prebaci u registar R2 vrednost sa mem
MULF R2, 60.0         # Pomnozi vrednost registra R2 sa konst
                      # i rezultat smesti u R2
MOVF R1, start_time   # Prebaci u registar R1 vrednost sa mem
                      # lokacije start_time
ADDF R1, R2           # Saberi vrednosti registara R1 i R2, r
MOVF cur_time, R1     # Prebaci na memorijsku lokaciju curr_t
```