Rješenja zadataka za vježbu iz predmeta Prevođenje programskih jezika

izrađeno ak. god. 2023./2024.

Sadržaj

1 Uvod

Zadatak 1
Navedite prednosti i nedostatke uporabe jezičnih procesora.

Prednosti primjene korisničkog jezika su približiti jezik programiranja području primjene, osloboditi jezik programiranja zavisnosti o arhitekturi računala, lakše učenje pravila jezika, lakši razvoj i razumijevanje programa, lakše otklanjanje pogrešaka i dokumentiranje, prenosivost, skraćeno vrijeme rješavanja problema. Nedostaci primjene korisničkog jezika su produljenje vremena potrebnog za prevođenje korisničkog programa u izvodivi strojni program, neučinkovit strojni program, nemoguće iskoristiti sve posebnosti građe računala, potrebno izgraditi razvojnu okolinu koja omogućuje ispitivanje i nadzor izvođenja korisničkih programa.

Zadatak 2
Navedite tri jezika koji su vezani uz definiciju jezičnog procesora.

Tri jezika koji ga definiraju su izvorni jezik L_i (jezik kojim je napisan izvorni program), ciljni jezik L_c (jezik kojim je napisan program nakog prevođenja) i jezik izgradnje L_g (jezik kojim je napisan jezični procesor): $\mathrm{JP}_{L_g}^{L_i \to L_c}$.

Zadatak 3
Navedite koja se pravila susreću kod programskih jezika te ih objasnite.

Leksička pravila → ime varijable mora počinjati slovom, ključne riječi su rezervirane i sl.
Sintaksna pravila → određuju dozvoljene strukture naredbi, dijelova programa i čitavog programa
Semantička pravila → interpretacijska pravila koja povezuju izvođenje izvornog programa s ponašanjem računala

Zadatak 4
Objasnite samoprevodioca.

Samoprevoditelj je jezični procesor koji je izgrađen primjenom izvornog jezika (izvorni jezik i jezik izgradnje su identični jezici), $\mathrm{JP}_{A}^{A \to B}$.

Zadatak 5
Objasnite cilj i osnovne korake postupka samopodizanja.

Cilj je prenjeti jezični procesor s računala a na računalo b koje je različite arhitekture. Imamo jezični procesor za računalo a: $\mathrm{JP}_{A}^{L \to A}$ i želimo izgraditi jezični procesor za računalo b: $\mathrm{JP}_{B}^{L \to B}$. Prvo izgradimo samoprevoditelj $\mathrm{JP}_{L}^{L \to B}$. Samoprevoditelj prevedemo koristeći $\mathrm{JP}_{A}^{L \to A}$ i dobivamo premosni prevoditelj $\mathrm{JP}_{A}^{L \to B}$. Pokretanjem premosnog prevoditelja na računalu a i prevođenjem samoprevoditelja $\mathrm{JP}_{L}^{L \to B}$ dobivamo jezični prevoditelj $\mathrm{JP}_{B}^{L \to B}$.

Zadatak 6
Objasnite što je funkcija preslikavanja jezičnog procesora i navedite njezine vrste.

Funkcija preslikavanja određuje koliko se naredbi izvornog programa prevodi u koliko naredbi ciljnog programa. Vrste preslikavanja su jedan u jedan, jedan u jedan ili jedan u više, jedan u više, više u jedan i složena funkcija preslikavanja (prevođenje naredbi jednog višeg programskog jezika u naredbe drugog višeg programskog jezika).

Zadatak 7
Objasnite podjelu jezičnih procesora s obzirom na broj prolazaka kroz izvorni program.

Jednoprolazni jezični procesori → nijedan korak rada ne sprema rezultat svog izvođenja izravno u memoriju računala
Višeprolazni jezični procesori → pojedini koraci rada spremaju rezultat izvođenja izravno u memoriju računala

Zadatak 8
Navedite i objasnite korake analize izvornog programa.

Faza analize rastavlja izvorni program u sastavne dijelove, provjerava pravila jezika, prijavljuje pogreške i zapisuje izvorni program primjenom različitih struktura podataka u memoriju računala. Analiza izvornog programa izvodi se u više zasebnih koraka:
leksička analiza = linearna analiza znakova izvornog programa, provjerava jesu li leksičke jedinke u izvornom programu ispravno napisane
sintaksna analiza = grupiranje leksičkih jedinki koje generira leksički analizator u hijerarhijske skupine sa zajedničkim značenjem
semantička analiza = most između faze analize i faze sinteze; provjera deklaracija varijabli, tipova podataka, indeksiranja polja, parametara programa i tijeka izvođenja programa

Zadatak 9
Objasnite razradbu jezičnih procesora s obzirom na dinamiku izvodenja procesa prevodenja.

Kompilatori → prevodi naredbe onim redoslijedom kojim su napisane u izvornom programu; izvođenje ciljnog programa započinje nakon što završi prevođenje

$$e_1^p \to e_2^p \to - - - \to e_N^p \to e_{x_1}^i \to e_{x_2}^i \to - - - \to e_{x_M}^i \to \text{STOP}$$

Interpretatori → ne prevodi cijeli izvorni program u ciljni program - prevede jednu naredbu i odmah ju izvede

$$e_{x_1}^p \to e_{x_1}^i \to e_{x_2}^p \to e_{x_2}^i \to - - - \to e_{x_M}^p \to e_{x_M}^i \to \text{STOP}$$

Dinamički interpretatori → paralelno izvodi procese prevođenja izvornog programa i izvođenja ciljnog programa; procesi su međusobno povezani FIFO spremnikom

Zadatak 10
Nabrojite i ukratko objasnite svaki od osnovnih koraka u izgradnji jezičnog procesora.

1. definicija jezičnog procesora → precizno određivanje pravila izvornog jezika, svojstva ciljnog jezika, svojstva arhitekture računala na kojem se izvodi jezični procesor i ciljni jezik, brzine rada, L_g itd.

2. strukturiranje jezičnog procesora → započinje prije nego završi definiranje; određuju se sučelja i osnovni koraci rada jezičnog procesora

3. programsko ostvarenje jezičnog procesora → izbor programskog jezika koji se koristi za ostvarenje jezičnog procesora, izbor odgovarajuće okoline za razvoj i ispitivanje te gradnja programa jezičnog procesora

4. ocjena → ispitati dosljednost, procijeniti koliko izgrađeni jezični procesor zadovoljava svojstva zadana definicijom

5. održavanje jezičnog procesora → nastoji se poboljšati učinkovitost rada, prilagodba novom području primjene, ispravljanje pogrešaka

Zadatak 11
Koristeći Hoareov sustav oznaka CSP, opišite vrste jezičnih procesora s obzirom na dinamiku izvođenja.

N ukupan broj naredbi u izvornom programu, e_i^p događaj prevođenja i-te naredbe izvornog programa, $e_{x_i}^i$ događaj izvođenja naredbe x_i izvornog programa, M broj izvedenih naredbi izvornog programa, >> komunikacija između procesa
PREVOĐENJE >> SPREMNIK^b >> IZVOĐENJE (opis rada dinamičkog interpretatora)

Zadatak 12
Objasnite kompilatore i interpretatore, odnosno njihovu razliku, ne koristeći Hoareov sustav oznaka CSP.

Interpretator, za razliku od kompilatora, ne prevodi cijeli izvorni program u ciljni program nego prevede jednu naredbu i odmah je izvede te se naredbe ne prevode onim redoslijedom kojim su navedene u izvornom programu već je redoslijed prevođenja naredbi određen redoslijedom njihova izvođenja.

Zadatak 13
Navedite faze rada jezičnog procesora od kojih se sastoje faza analize izvornog programa i faza sinteze ciljnog programa.

koraci analize: leksička, sintaksna i semantička analiza
koraci sinteze: generiranje međukôda, strojno nezavisno optimiranje, generiranje strojnog programa, strojno zavisno optimiranje, priprema strojnog programa za izvođenje

Zadatak 14

Za svako od tri računala A, B i C s odgovarajućim strojnim jezicima a, b i c izgradite jezične procesore koji prevode jezik l u jezik n i koji su izvedivi na pojedinim računalima. Jezične procesore treba izgraditi pomoću raspoloživih jezičnih procesora $\mathrm{JP}_l^{l\to n}$, $\mathrm{JP}_a^{l\to a}$, $\mathrm{JP}_b^{n\to o}$, $\mathrm{JP}_o^{o\to b}$ i $\mathrm{JP}_c^{o\to c}$

$$\mathrm{JP}_l^{l\to n} \xrightarrow{\mathrm{JP}_a^{l\to a}} \mathrm{JP}_a^{l\to n}$$

$$\mathrm{JP}_l^{l\to n} \xrightarrow{\mathrm{JP}_l^{l\to n}} \mathrm{JP}_n^{l\to n} \xrightarrow{\mathrm{JP}_b^{n\to o}} \mathrm{JP}_o^{l\to n} \xrightarrow{\mathrm{JP}_o^{o\to b}} \mathrm{JP}_b^{l\to n}$$

$$\mathrm{JP}_l^{l\to n} \xrightarrow{\mathrm{JP}_l^{l\to n}} \mathrm{JP}_n^{l\to n} \xrightarrow{\mathrm{JP}_b^{n\to o}} \mathrm{JP}_o^{l\to n} \xrightarrow{\mathrm{JP}_c^{o\to c}} \mathrm{JP}_c^{l\to n}$$

Zadatak 15

Za računalo A postoji gotov jezični procesor $\mathrm{JP}_a^{l\to a}$, dok je na računalu B na raspolaganju jezični procesor $\mathrm{JP}_b^{n\to c}$. Računalo C novo je računalo i za njega ne postoji nikakav jezični procesor, a potrebno je program napisan u jeziku m prevesti u ciljni program za to računalo. Raspoloživ je i jezični procesor $\mathrm{JP}_m^{m\to n}$, a zbog modularnosti je potrebno napisati jezični procesor $\mathrm{JP}_?^{m\to l}$. Odredite u kojem je višem jeziku (l, m ili n) potrebno napisati taj jezični procesor da bi se na najkraći mogući način preveo program te navedite najkraći postupak prevodenja programa iz jezika m u ciljni jezik c.

Zadatak 16

Tijekom razvoja složenog sustava koji se sastoji od tri računalne arhitekture A, B i C pojavila se potreba za razvojem jezičnog procesora $\mathrm{JP}_b^{o\to b}$. Za potrebe sustava već su napisani jezični procesori $\mathrm{JP}_a^{l\to a}$, $\mathrm{JP}_c^{l\to n}$ i $\mathrm{JP}_c^{o\to b}$. Osim toga, preko Interneta su javno dostupna još tri jezična procesora: $\mathrm{JP}_l^{m\to b}$, $\mathrm{JP}_m^{n\to o}$ i $\mathrm{JP}_n^{o\to b}$. Budući da se projekt približava krajnjem roku, nema dovoljno vremena za pisanje novog jezičnog procesora pa je potrebno jezični procesor $\mathrm{JP}_b^{o\to b}$ konstruirati pomoću raspoloživih jezičnih procesora. Napišite postupak konstrukcije.

$$\mathrm{JP}_n^{o\to b} \xrightarrow{\mathrm{JP}_m^{n\to o}} \mathrm{JP}_o^{o\to b} \xrightarrow{\mathrm{JP}_c^{o\to b}} \mathrm{JP}_b^{o\to b}$$

Zadatak 17

Za računalo A postoji jezični procesor $\mathrm{JP}_a^{z\to x}$, dok je na računalu B dostupan jezični procesor $\mathrm{JP}_b^{x\to a}$. Raspoloživ je i jezični procesor $\mathrm{JP}_z^{x\to y}$. Odredite u kojem višem programskom jeziku (x, y ili z) treba izgraditi jezični procesor $\mathrm{JP}_?^{y\to b}$, tako da se može ostvariti prevodenje programa napisanog u jeziku y u ciljni jezik b. Navedite sve korake u postupku prevodenja programa.

izgradnja dodatnih jezičnih procesora:

$$\mathrm{JP}_z^{x\to y} \xrightarrow{\mathrm{JP}_a^{z\to x}} \mathrm{JP}_x^{x\to y}$$

$$\mathrm{JP}_x^{x\to y} \xrightarrow{\mathrm{JP}_b^{x\to a}} \mathrm{JP}_a^{x\to y}$$

ako je jezik izgradnje x:

$$\mathrm{JP}_x^{y\to b} \xrightarrow{\mathrm{JP}_b^{x\to a}} \mathrm{JP}_a^{y\to b}$$

ako je jezik izgradnje z:

$$\mathrm{JP}_z^{y\to b} \xrightarrow{\mathrm{JP}_a^{z\to x}} \mathrm{JP}_x^{y\to b} \xrightarrow{\mathrm{JP}_b^{x\to a}} \mathrm{JP}_a^{y\to b}$$

Za jezik izgradnje y nije moguće izgraditi traženi jezični procesor (nemamo na raspolaganju jezični procesor koji bi jezik izgradnje y preveo u neki od viših programskih jezika, u jezik a ili u jezik b). Zaključak je da je potrebno JP izgraditi u jeziku x jer se korištenjem tog jezika u najmanjem broju koraka dolazi do željenog jezičnog procesora.

Zadatak 18

Računalni sustav sastoji se od jednog računala arhitekture A i jednog računala arhitekture B. Za računalnu arhitekturu B razvijen je jezični procesor $\mathrm{JP}_b^{c\to b}$ kojim je omogućeno izvođenje programa napisanih višim programskim jezikom c na računalu arhitekture B. Osim toga, na raspolaganju su jezični procesori $\mathrm{JP}_p^{c\to b}$, $\mathrm{JP}_p^{p\to b}$, $\mathrm{JP}_c^{p\to a}$ i $\mathrm{JP}_c^{c\to b}$. Uporabom navedenih jezičnih procesora izgradite jezični procesor koji će na računalu arhitekture A omogućiti prevođenje programa napisanih višim programskim jezikom p u programe izvodive na računalu arhitekture B.

$$\mathrm{JP}_c^{p\to a} \xrightarrow{\mathrm{JP}_b^{c\to b}} \mathrm{JP}_b^{p\to a}$$

$$\mathrm{JP}_p^{p\to b} \xrightarrow{\mathrm{JP}_b^{p\to a}} \mathrm{JP}_a^{p\to b}$$

Zadatak 19

Prikažite postupak izgradnje izvodivog jezičnog procesora koji prevodi jezik l u strojni jezik a. Na raspolaganju su samoprevodilac koji prevodi jezik l u strojni jezik a, $\mathrm{JP}_a^{p\to s}$, $\mathrm{JP}_a^{s\to a}$, $\mathrm{JP}_b^{l\to q}$, $\mathrm{JP}_b^{r\to a}$ i $\mathrm{JP}_r^{q\to p}$ te računala A i B na kojima se mogu izvoditi programi pisani strojnim jezicima a i b.

$$\mathrm{JP}_l^{l\to a} \xrightarrow{\mathrm{JP}_b^{l\to q}} \mathrm{JP}_q^{l\to a} \xrightarrow{\mathrm{JP}_r^{q\to p}} \mathrm{JP}_p^{l\to a} \xrightarrow{\mathrm{JP}_a^{p\to s}} \mathrm{JP}_s^{l\to a} \xrightarrow{\mathrm{JP}_a^{s\to a}} \mathrm{JP}_a^{l\to a}$$

Zadatak 20

Tijekom razvoja složenog sustava koji se sastoji od dvije računalne arhitekture A i B pojavila se potreba za razvojem jezičnog procesora $\mathrm{JP}_b^{l\to a}$. Za potrebe sustava već je napisan jezični procesor $\mathrm{JP}_a^{m\to a}$. Osim toga, preko Interneta su javno dostupna još četiri jezična procesora: $\mathrm{JP}_m^{l\to b}$, $\mathrm{JP}_l^{m\to a}$, $\mathrm{JP}_l^{l\to a}$ i $\mathrm{JP}_m^{m\to b}$. Budući da se projekt približava krajnjem roku, nema dovoljno vremena za pisanje novog jezičnog procesora pa je potrebno jezični procesor $\mathrm{JP}_b^{l\to a}$ konstruirati pomoću raspoloživih jezičnih procesora. Napišite postupak konstrukcije $\mathrm{JP}_b^{l\to a}$.

$$\mathrm{JP}_{l}^{l\to a} \xrightarrow{\mathrm{JP}_{m}^{l\to b}} \mathrm{JP}_{b}^{l\to a}$$

Zadatak 21

Za novo računalo "Amiga 2001" (računalo C) kao osnovni jezik treba upotrijebiti "C++" (jezik m) pa je potrebno izgraditi jezični procesor $\mathrm{JP}_{c}^{m\to c}$. Na "PC" računalu (računalo B) postoji jezični procesor $\mathrm{JP}_{b}^{l\to b}$ koji prevodi jezik "C" (jezik l) u Intelov strojni jezik te jezični procesor $\mathrm{JP}_{b}^{k\to c}$, koji prevodi jezik "BCPL" (jezik k) u strojni jezik računala "Amiga 2001". Na raspolaganju je i računalo "Amiga 4000" (računalo A) sa pripadnim jezičnim procesorom $\mathrm{JP}_{a}^{k\to a}$, koji prevodi jezik "BCPL" u Motorolin strojni jezik. Pored ovoga postoje i jezični procesori $\mathrm{JP}_{k}^{l\to k}$ (kros-kompilator) koji prevodi jezik "C" u jezik "BCPL" i koji je napisan u "BCPL"-u te $\mathrm{JP}_{l}^{m\to c}$ koji prevodi "C++" u strojni jezik računala "Amiga 2001", a napisan je u "C"-u. Napišite postupak samopodizanja kojim se može dobiti traženi jezični procesor uz uporabu postojećih jezičnih procesora.

$$\mathrm{JP}_{l}^{m\to c} \xrightarrow{\mathrm{JP}_{k}^{l\to k}} \mathrm{JP}_{k}^{m\to c} \xrightarrow{\mathrm{JP}_{b}^{k\to c}} \mathrm{JP}_{c}^{m\to c}$$

Zadatak 22

U računalnom sustavu koriste se tri računala A, B i C, različitih arhitektura, pri čemu računalo A izvodi strojni jezik a, računalo B izvodi strojni jezik b, a računalo C izvodi strojni jezik c. Za računalo C dostupni su $\mathrm{JP}_{c}^{y\to c}$ i $\mathrm{JP}_{c}^{z\to c}$ te samoprevodioci $\mathrm{JP}_{y}^{y\to a}$ i $\mathrm{JP}_{z}^{z\to b}$. Osim toga, dostupni su jezični procesori $\mathrm{JP}_{y}^{a\to y}$, $\mathrm{JP}_{z}^{y\to c}$ i $\mathrm{JP}_{a}^{x\to c}$. Navedite postupak konstrukcije $\mathrm{JP}_{c}^{x\to c}$ primjenom isključivo raspoloživih jezičnih procesora. Dopušteno je stvaranje i novih jezičnih procesora, ali isključivo prevođenjem postojećih jezičnih procesora pomoću drugih raspoloživih jezičnih procesora.

$$\mathrm{JP}_{a}^{x\to c} \xrightarrow{\mathrm{JP}_{y}^{a\to y}} \mathrm{JP}_{y}^{x\to c} \xrightarrow{\mathrm{JP}_{z}^{y\to c}} \mathrm{JP}_{c}^{x\to c}$$

2 Leksička analiza

Zadatak 23
Opišite dinamiku izvođenja leksičke analize.

Suradnja leksičkog i sintaksnog analizatora ostvaruje se na dva načina. Ako se ostvari putem poziva potprograma, onda je leksički analizator potprogram sintaksnog analizatora. Ako se suradnja ostvari razmjenom čitave tablice uniformnih znakova, onda je leksički analizator zasebni prolaz jezičnog procesora.

Zadatak 24
Opišite program simulator leksičkog analizatora zasnovan na tablici prijelaza ε-NKA.

1. u program simulator ugradi se tablica prijelaza ε-NKA $M = (Q, \Sigma, \delta, p_0, F)$ koji je izgrađen na osnovi regularnih definicija klasa leksičkih jedinki
2. u ulazni spremnik učita se cijeli izvorni program
3. neka je R skup stanja ε-NKA; na početku rada skup stanja je $R = \varepsilon$-okruženje(p_0)
4. čita znak (a) po znak i određuje $R = \varepsilon$-okruženje$(\delta(Q, a))$; izabire onaj q_i iz skupa $P = R \cap F$ kojemu je pridružen regularni izraz r_i koji je naveden prije svih ostalih regularnih izraza koji su pridruženi ostalim stanjima u skupu P; kad $R = \{\}$ (prazan skup) prekida se čitanje
5. nakon što skup stanja R postane prazan, na temelju vrijednosti varijable Izlaz određuje se leksička jedinka i klasa te jedinke
6. ima li još znakova u ulaznom spremnik koji nisu analizirani, nastavlja se korakom 4, inače završava rad simulatora

Zadatak 25
Opišite način izrade globalne tablice znakova.

Leksički analizator odredi klasu leksičke jedinke, zapiše uniformni znak u tablicu uniformnih znakova i započinje pretraživanje ostalih tablica. Ako za leksičku jedinku postoji zapis u odgovarajućoj tablici (npr. u tablici identifikatora za identifikatore), onda se u tablicu uniformnih znakova zapisuje kazaljka koja pokazuje na mjesto pronađenog zapisa. Ako ne postoji traženi zapis u odgovarajućoj tablici, stvara se novi zapis i u tablicu uniformnih znakova upisuje se kazaljka koja pokazuje na novostvoreni zapis. Za konstante se u tablicu konstanti zapisuje i tip konstante, a za KROS jedinke zapisuju se dodatni parametri (npr. podaci o prednostima operatora, je li znak prekidni i sl.).

Zadatak 26
Nabrojite i objasnite osnovne klase leksičkih jedinki.

Specijalni znakovi (zagrade, zarezi, točka), operatori (za zbrajanje, oduzimanje, množenje), ključne riječi (ako, onda, inače), identifikatori (imena varijabli, potprograma), konstante (cjelobrojne, znakovne)

Zadatak 27
Opišite program simulator leksičkog analizatora zasnovan na tablici prijelaza DKA.

Vidi zadatak 40.

Zadatak 28
Objasnite kako leksički analizator utvrđuje leksičku pogrešku i navedite dva postupka oporavka od pogreške.

Od pogreške se oporavlja odbacivanjem krajnje lijevog znaka niza koji nema nijedan prefiks definiran barem jednim od regularnih izraza. Nakon odbacivanja krajnje lijevog znaka, taj se znak ispisuje i nastavlja se s analizom ostatka niza.

Zadatak 30
Opišite podatkovne strukture leksičkog analizatora.

Podatkovnu strukturu leksičkog analizatora čine izvorni program, tablica uniformnih znakova i tablica znakova.
tablica uniformnih znakova → zapisani uniformni znakovi onim redoslijedom kojim su leksičke jedinke zadane u izvornom programu; prva je kazaljka sam uniformni znak (pokazuje na jednu od tri tablice), a druga kazaljka pokazuje na mjesto zapisa leksičke jedinke u tablici određenoj prvom kazaljkom
tablica znakova → razlaže se na tablicu identifikatora, tablicu konstanti i KROS tablicu

Zadatak 31
Objasnite metode opisa leksičkih jedinki, pravila određivanja klasa leksičkih jedinki i postupak grupiranja leksičkih jedinki koje se koriste za izgradnju generatora leksičkog analizatora.

Leksička se pravila zadaju primjenom regularnih izraza i regularnih definicija. Za svaku klasu leksičkih jedinki definira se zasebni regularni izraz, ako je niz definiran primjenom više regularnih izraza uzima se onaj koji je prvi zapisan u nizu regularnih izraza. Tijekom grupiranja leksičkih jedinki, određuje se najdulji prefiks niza koji je definiran barem jednim regularnim izrazom.

Zadatak 32
Navedite i ukratko objasnite pet zadataka leksičkog analizatora.

1. Slijedno čita tekst izvornog programa znak po znak = jedini korak rada koji izravno pristupa znakovima teksta izvornog programa
2. Stvara učinkovit zapis znakova izvornog programa = kôdiranje znakova izvornog programa te nije potrebno razlikovati sve znakove
3. Odbacuje znakove koji se ne koriste u daljnjim koracima rada jezičnog procesora = izbacivanje kometara, bjelina, tabulatora, CR, LF i sl.

4. Grupira znakove u leksičke jedinke = leksičke jedinke zapisuje odvojene prekidnim znakovima ili slobodnim načinom zapisa

5. Odrediti klase leksičkim jedinkama = središnji algoritam leksičkog analizatora

6. Provjera leksičkih pravila = pravila klasa zadana su regularnim izrazima; pokušava leksičku jedinku svrstati u jednu od klasa

7. Pronalazi pogreške i određuje mjesto pogreške u izvornom programu

8. Zapisuje parametre leksičkih jedinki u tablicu znakova = u tablicu se sprema leksička jedinka u obliku u kakvom je u izvornom programu i njezini parametri

9. Čuva tekstualnu strukturu izvornog programa

Zadatak 33
Navedite i na primjeru objasnite dva osnovna načina razrješavanja nejednoznačnosti u leksičkoj analizi.

Prvi je način pretraživanje desnog konteksta - npr. u programskom jeziku FORTRAN niz *DO* ključna je riječ ako i samo ako iza niza *DO* u ulaznom spremniku slijedi niz precizno definiran regularnim izrazom. Drugi je način pretraživanje lijevog konteksta - npr. operatori + i − unarni su operatori u slučaju kad se neposredno ispred njih nalazi znak pridruživanja.

Zadatak 34
Navedite i objasnite varijable koje koristi simulator zasnovan na tablici prijelaza DKA. U ovisnosti o navedenim varijablama, objasnite postupak simulatora za grupiranje i određivanje klase leksičke jedinke.

Varijable koje koristi simulator DKA su početak (pokazuje na početak neanaliziranog dijela niza), završetak (pokazuje na posljednji pročitani znak), posljednji (pokazuje na posljednji znak najduljeg prepoznatog prefiksa niza) i izraz (poprima vrijednost oznake regularnog izraza najduljeg prepoznatog prefiksa). Početno je stanje varijabli završetak i izraz nula, a početak i posljednji imaju vrijednost jedan. Nakon što DKA prijeđe u stanje $\varnothing$, na temelju vrijednosti varijable izraz određuje se grupiranje u leksičku jedinku i njezina klasa. Ukoliko je vrijednost varijable izlaz nula, simulator nije pronašao nijedan prefiks koji je definiran barem jednim regularnim izrazom i pokreće se postupak oporavka od pogreške odbacivanjem krajnje lijevog znaka.

Zadatak 35
Objasnite postupak razrješenja nejednoznačnosti u leksičkoj analizi pretraživanjem lijevog konteksta.

Za potrebe pretraživanja lijevog konteksta definiraju se dodatna stanja. Ulazak i izlazak iz dodatnih stanja zadaje se u akcijama pridruženim pojedinim regularnim izrazima. Lijeva kontekstna ovisnost zadana je ovako: `<ime stanja> r`.

Zadatak 36

Općenito objasnite (ne na primjeru) postupak primjene pretraživanja desnog konteksta za razrješavanje nejednoznačnosti u leksičkoj analizi. Potrebno je definirati kako se u regularnim izrazima zadaje desni kontekst, kako se na osnovi regulranih izraza sa zadanim desnim kontekstom stvara konačni automat te kako se dobiveni konačni automat koristi za leksičku analizu.

Pretraživanje desnog konteksta zadaje se na sljedeći način: `r/r'`. U postupku izgradnje ε-NKA na temelju regularnih izraza `r/r'` definira se regularni izraz `rr'`. Tijekom rada program koristi regularne izraze `r` i `rr'`. Pronađe li simulator prefiks niza znakova koji je definiran regularnim izrazom `rr'`, grupiraju se samo znakovi koji su definirani regularnim izrazom `r`. Analiza se nastavlja krajnje lijevim znakom u nizu definiranom regularnim izrazom `r'`.

Zadatak 37

Navedite pravila za određivanje klase leksičke jedinke i grupiranje znakova u leksičke jedinke.

P1 Za sve leksičke jedinke koje je potrebno razlikovati tijekom sintaksne analize definira se zasebni regularni izraz.
P2 Ako je niz znakova x definiran primjenom dva regularna izraza r_k i r_l koji označavaju dvije različite klase leksičkih jedinki, k i l, onda je niz x u klasi k ako i samo ako je regularni izraz r_k zapisan u listi regularnih izraza prije izraza r_l.
P3 Tijekom grupiranja traži se najdulji prefiks niza w koji je definiran barem jednim regularnim izrazom. Neka je w niz znakova izvornog programa, a x i y su nizovi znakova definirani zadanim regularnim nizovima. Niz znakova x, koji je prefiks niza w, jest leksička jedinka ako i samo ako bilo koji drugi prefiks y niza w jest ujedno i prefiks niza x.

Zadatak 38

Općenito definirajte (ne na primjeru) ulaze i izlaze iz programa generatora leksičkog analizatora i programa leksičkog analizatora ako je leksički analizator ostvaren kao zasebni prolaz jezičnog procesora.

generator leksičkog analizatora: ulaz je opis procesa leksičkog analizatora (regularni izrazi, imena leksičkih jedinki i sl.), izlaz je izvorni kôd programa leksičkog analizatora
leksički analizator: ulaz je niz znakova izvornog programa, izlaz je tablica uniformnih znakova

Zadatak 39

Navedite strukture podataka pogodne za ostvarenje tablice znakova i asimptotsku složenost osnovnih operacija nad tablicom znakova za svaku predloženu strukturu podataka.

linearna lista → jedna kazaljka pokazuje na početak, druga na kraj; traženje $O(n)$, dodavanje $O(1)$
uređena lista → pogodna za KROS tablicu jer se ne mijenja; traženje $O(\log_2 n)$, dodavanje $O(n)$
binarno stablo → traženje $O(\log_2 n)$, dodavanje $O(\log_2 n)$
raspršeno adresiranje → u idealnom slučaju obje operacije $O(1)$, ako imamo c različitih ključeva i n zapisa, složenost pretraživanja proporcionalna je $\frac{n}{2c}$

Zadatak 40

Opišite algoritam leksičkog analizatora zasnovanog na tablici prijelaza DKA.

1. u program simulator ugradi se tablica prijelaza DKA $M' = (Q', \Sigma', \delta', p_0', F')$ koji se izgradi na temelju ε-NKA $M = (Q, \Sigma, \delta, p_0, F)$; dodaje se i neprihvatljivo stanje $\varnothing$ u koje se prelazi nema li ε-NKA definiranih prijelaza

2. u ulazni spremnik spremi se cijeli izvorni program

3. simulator započinje rad čitanjem krajnje lijevog znaka; koriste se kazaljke početak (pokazuje na početak neanaliziranog dijela niza), završetak (pokazuje na posljednji pročitani znak), posljednji (pokazuje na posljednji znak najduljeg prepoznatog prefiksa niza) i izraz (poprima vrijednost oznake regularnog izraza najduljeg prepoznatog prefiksa)

4. čita znak (a) po znak i određuje $q' = \delta'(q', a)$; izabire onaj q_i iz skupa $P = \{q_0, q_1, ..., q_k\} \cap F$ kojemu je pridružen regularni izraz r_i; kad $q' = \varnothing$ prekida se čitanje

5. nakon što DKA prijeđe u stanje $\varnothing$, na temelju vrijednosti varijable Izlaz određuje se leksička jedinka i klasa te jedinke

6. ima li još znakova u ulaznom spremnik koji nisu analizirani, nastavlja se korakom 4, inače završava rad simulatora

Zadatak 41

Za zadani program nacrtajte sve tablice koje se stvaraju u leksičkoj analizi.

```
program Phoebe;
   j:=1; k:=1;
   ispisi (j,k);
   za i:=3 do 20 cini
       k:=j+k;
       j:=k-j;
       ispisi (k);
   kraj
kraj
```

KROS	1
IDN	1
KROS	2
IDN	2
KROS	3
KON	1
KROS	2
IDN	3
KROS	3
KON	1
KROS	2
IDN	4
KROS	4
IDN	2
KROS	6
IDN	3
KROS	5
KROS	2
KROS	7
IDN	5
KROS	3
KON	2
KROS	8
KON	3
KROS	9

IDN	3
KROS	3
IDN	2
KROS	10
IDN	3
KROS	2
IDN	2
KROS	3
IDN	3
KROS	11
IDN	2
KROS	2
IDN	4
KROS	4
IDN	3
KROS	5
KROS	2
KROS	12
KROS	12

1	program
2	;
3	=
4	(
5	)
6	,
7	za
8	do
9	cini
10	+
11	−
12	kraj

1	Phoebe
2	j
3	k
4	ispisi
5	i

1	1
2	3
3	20

Zadatak 52

Ostvaren je program simulator leksičkog analizatora zasnovan na tablici prijelaza DKA s jednostavnim postupkom oporavka od pogreške. Simulator prepoznaje dva niza: *AUTO* i *AUTOMOBIL*. Na ulazu automata pojavljuje se niz *AUTOMATSKIAUTOMOBIL*. Odredite koje će nizove simulator leksičkog analizatora prepoznati i hoće li ispisati neke greške. Potrebno je i ispisati ablicu stanja unutarnjih kazaljki (početak, završetak i posljednji) programa simulatora za svaki učitani znak.

početak	završetak	posljednji	izraz	najdulji poznati prefiks
1	0	1	0	-
1	1	1	0	-
1	2	1	r_x	A
1	3	2	r_x	AU
1	4	3	r_x	AUT
1	5	4	r_x	$AUTO$
5	5	5	0	-
5	6	5	0	M
5	7	6	0	MA
⋮	⋮	⋮	⋮	⋮
5	20	19	0	$MATSKIAUTOMOBIL$
6	6	6	0	-
6	7	6	r_x	A
6	8	7	0	AT
⋮	⋮	⋮	⋮	⋮
6	20	19	0	$ATSKIAUTOMOBIL$
7	7	7	0	-
7	8	7	0	T
7	9	8	0	TS
⋮	⋮	⋮	⋮	⋮
7	20	19	0	$TSKIAUTOMOBIL$
⋮	⋮	⋮	⋮	⋮
11	12	11	r_x	A
⋮	⋮	⋮	⋮	⋮
11	20	19	r_x	$AUTOMOBIL$

Zadatak 56

Zadan je jezik L koji sadrži cjelobrojne aritmetičke izraze. Leksičke su jedinke jezika L varijable, konstante, binarni operatori $\{+, -, *, /\}$, okrugle zagrade i unarni operatori $\{-, ++, --\}$. Varijable se sastoje od slova i brojki te moraju započinjati slovom, a konstante su cjelobrojne. Unarni operator $-$ označava negaciju varijable ili konstante. Unarni operatori $++$ i $--$ imaju značenje kao u jeziku C. Definirajte pravila leksičkog analizatora kojima se ulazni niz rastavlja na leksičke jedinke.

r1	$(a+b+\cdots+z)\,[(a+b+\cdots+z)+(0+1+\cdots+9)]^*$	varijable
r2	$(0+1+\cdots+9)^+$	konstante
r3	$(\ \vert\ )\ \vert\ +\ \vert\ -\ \vert\ *\ \vert\ /$	zagrade i binarni operatori
r4	$=\ -$	uđi u stanje UNARNI; ODBACI
r5	$=(++\ \vert\ --)$	uđi u stanje UNARNI; ODBACI
r6	<UNARNI> $-(a+b+\cdots+z)\,[(a+b+\cdots+z)+(0+1+\cdots+9)]^*$	izađi iz stanja UNARNI
r7	<UNARNI> $-(0+1+\cdots+9)^+$	izađi iz stanja UNARNI
r8	<UNARNI> $(++\ \vert\ --)(a+b+\cdots+z)\,[(a+b+\cdots+z)+(0+1+\cdots+9)]^*$	izađi iz stanja UNARNI
r9	<UNARNI> $(++\ \vert\ --)\,(0+1+\cdots+9)^+$	izađi iz stanja UNARNI

Zadatak 57

U postupku sintaksne analize programskog jezika X potrebno je razlikovati dekadske, oktalne i heksadekadske pozitivne cjelobrojne konstante. Napišite regularne izraze koji će omogućiti ispravno određivanje klase cjelobrojne konstante u leksičkoj analizi. Oktalne konstante započinju znamenkom 0 (npr. 0134, 071 i 00032). Dopušteno je da dekadske konstante započinju vodećim nulama ako sadrže barem jednu znamenku 8 ili 9 (dekadske su konstante npr. 00039, 488 i 455). Heksadekadske konstante započinju nizom 0x i dalje sadrže dekadske znamenke i mala slova a, b, c, d, e, f (npr. 0x13a, 0x043f i 0x00fed).

oktalne: $0(0+1+2+\cdots+7)^+$

dekadske: $[0^+(1+2+\cdots+7)^*(8+9)^+(1+2+\cdots+9)^*]+(1+2+\cdots+9)(0+1+2+\cdots+9)^*$

heksadekadske: $0x(0+1+2+\cdots+9+a+\cdots+f)(0+1+2+\cdots+9+a+\cdots+f)^*$

Zadatak 58

Prikažite postupak obrade i izlaz leksičkog analizatora zasnovanog na regularnim izrazima iz tablice na sljedećim ulaznim nizovima (obrada svakog niza je nezavisna): *aabab*, *ababbba* i *abababc*.

r1	$aab(c)*$	ispiši(”r1”)
r2	$(a)^*b$	ispiši(”r2”)
r3	$abab$	ispiši(”r3”)
r4	ab/c	ispiši(”r4”)
r5	$ababb$	uđi u stanje S; ODBACI;
r6	bbb	ispiši("r6")
r7	$< S > bba$	ispiši("r7"); izađi iz stanja S;
r8	$(c)*$	ispiši("r8")

aabab → r1 (zadovoljava i r2, ali je r1 napisan prije), r2
ababbba → r3, r7 (r5 je osigurao promjenu stanja u stanje S, ali zbog ONEMOGUĆI se ne primjenjuje)
abababc → r3, r2, r8 (r2 ima prednost pred r4)

Zadatak 59

Prikažite postupak obrade i izlaz leksičkog analizatora zasnovanog na regularnim izrazima iz tablice (ista kao i tablica u zadatku 58) na sljedećim ulaznim nizovima (obrada svakog niza je nezavisna): *aababcaaab* i *ccababbba*.

aababcaaab → r1 (*aab*), r2 (*ab*), r8 (*c*), r2 (*aaab*)
ccababbba → r8 (*cc*), r3 (*abab*), r7 (*bba*)

Zadatak 60

Prikažite postupak obrade i izlaz leksičkog analizatora zasnovanog na regularnim izrazima iz tablice na sljedećim ulaznim nizovima (obrada svakog niza je nezavisna): *aab*, *aaa%%b* i *b#%%*.

r1	$\%\%b$	ispiši(”r1”)
r2	$\%\%ba^*$	ispiši(”r2”)
r3	a	ispiši(”r3”)
r4	aa/b	ispiši(”r4”)
r5	aaa	ispiši("r5")
r6	b	ispiši("r6")
r7	$b(\%\|\#)$	uđi u stanje S; ODBACI
r8	$< S > \#\%\%$	ispiši("r8") izađi iz stanja S

aab → r4 (*aa*), r6 (*b*)
aaa%%b → r5 (*aaa*), r1 (*%%b*)
b#%% → r6 (*b*), r8 (*#%%*)

Zadatak 61

Na osnovi navedenih pravila odredite i objasnite izlaz leksičkog analizatora za nizove $yyy++x$, yyx i $x!++$.

r1	$++x$	ispiši(”r1”)
r2	$++xy^*$	ispiši(”r2”)
r3	y	ispiši(”r3”)
r4	yy/x	ispiši(”r4”)
r5	yyy	ispiši("r5")
r6	x	ispiši("r6")
r7	$x(+\|!)$	uđi u stanje S; ODBACI
r8	$< S >!++$	ispiši("r8") izađi iz stanja S

$yyy++x \rightarrow$ r5 (yyy), r1 $(++x)$
$yyx \rightarrow$ r4 (yy), r6 (x)
$x!++ \rightarrow$ r6 (x), r8 $(!++)$

Zadatak 62

Prikažite postupak obrade i izlaz leksičkog analizatora zasnovanog na regularnim izrazima na sljedećim ulaznim nizovima: $sedam78ggg$ i $gggosam8sedam7devet8$.

r1	$sedam\|7$
r2	$osam\|8$
r3	$devet\|9$
r4	$(a\|b\|\ldots\|z)^*(0\|1\|\ldots\|9)^*$

$sedam78ggg \rightarrow$ r4 $(sedam78)$, r4 (ggg)
$gggosam8sedam7devet8 \rightarrow$ r4 $(gggosam8)$, r4 $(sedam7)$, r4 $(devet8)$

3 Sintaksna analiza

Zadatak 65
Definirajte relacije *IspodZnaka* i *ReduciranZnakom* za parsiranje tehnikom *Pomakni-Pronađi*.

Za znakove A i x vrijedi relacija $IspodZnaka(A, x)$ ako i samo ako je ispunjen jedan od sljedećih uvjeta:

1. Znak A je izravo ispred znaka B na desnoj strani barem jedne produkcije gramatike, $IzravnoIspredZnaka(A, B)$, a znak x započinje barem jedan niz generiran iz B, $x \in \text{ZAPOČINJE}(B)$
2. A je oznaka dna stoga ∇ i $x \in \text{ZAPOČINJE}(S)$, S početni nezavršni znak gramatike

Za znakove A i x vrijedi relacija $ReduciranZnakom(A, x)$ ako i samo ako je ispunjen jedan od sljedećih uvjeta:

1. Znak A krajnje je desni znak produkcije $L \to \alpha\, A$, a znak x slijedi znak L u barem jednom nizu generiranom iz početnog nezavršnog znaka gramatike S tj. $x \in \text{SLIJEDI}(L)$
2. A je početni nezavršni znak gramatike, S, a x je oznaka kraja niza, $\perp$

Zadatak 66
Navedite uvjete pod kojima je kontekstno neovisna gramatika ujedno i S-gramatika.

1. Desna strana bilo koje produkcije započinje završnim znakom gramatike.
2. Desna strana niti jedne produkcije nije prazni niz ε.
3. Ako više produkcija ima isto nezavršni znak na lijevoj strani, onda desne strane tih produkcija započinju različitim završnim znakovima.

Zadatak 67
Navedite uvjete pod kojima je kontekstno neovisna gramatika ujedno i $LL(1)$-gramatika.

Ako više produkcija ima isti nezavršni znak na lijevoj strani, onda njihovi skupovi PRIMIJENI nemaju zajedničkih elemenata.

Zadatak 69
Objasnite postupak određivanja relacija prednosti na temelju zadane gramatike.

- $l \lessdot d$ ako je na desnoj strani produkcije znak l neposredno ispred nezavršnog znaka C, $A \to \cdots l\, C \cdots$ i ako C generira međuniz u kojem je krajnje lijevi završni znak d
- $l \doteq d$ ako je na desnoj strani produkcije završni znak l neposredno ispred završnog znaka d, $A \to \cdots l\, d \cdots$ ili je između njih samo jedan nezavršni znak, $A \to \cdots l\, B\, d \cdots$
- $l \gtrdot d$ ako je na desnoj strani produkcije nezavršni znak B neposredno ispred završnog znaka d, $A \to \cdots B\, d \cdots$ i ako nezavršni znak B generira međuniz u kojemu je krajnje desni završni znak l

Zadatak 70

U pseudokodu sličnom jeziku C napišite algoritam parsiranja tehnikom prednosti operatora.

Postavi KAZALJKU da pokazuje na krajnje lijevi znak niza $w\bot$
```
dok(1) {
    ako ZnakNaVrhuStoga == ∇ && Ulaz == ⊥
        Prihvati();
    inače {
        ako (x ⇐ y || x ⇔ y) {
            StaviNaStog(y);
            Pomakni KAZALJKU na sljedeći znak ulaznog niza;
        } inače ako (x ⇒ y)) {
            UzmiSVrhaStoga(1 znak);
            dok !(ZnakNaVrhuStoga ⇐ UzetiZnak)
                UzmiSVrhaStoga(1 znak);
        }
        inače
            Odbaci();
    }
}
```

Zadatak 71

Navedite i kratko opišite postupke pretvorbe produkcija u produkcije $LL(1)$-gramatike.

1. Lijevo izlučivanje - vidi zadatak 72
2. Uklanjanje lijeve rekurzije - vidi zadatak 73
3. Zamjena nezavršnih znakova - neka gramatika ima n produkcija oblika $A \to \alpha_\mathrm{i}$, i $= 1, 2, \ldots, n$ i zadana je produkcija oblika $B \to \beta\, A\, \gamma \Rightarrow$ tih $n+1$ produkcija zamijeni se sa sljedećih n produkcija $B \to \beta\, \alpha_\mathrm{i}\, \gamma$, i $= 1, 2, \ldots, n$

Zadatak 72

Općenito definirati postupak lijevog izlučivanja koji se koristi pri pretvorbi produkcija u produkcije $LL(1)$-gramatike.

Neka gramatika ima n produkcija oblika $A \to \alpha\, \beta_\mathrm{i}$, i $= 1, 2, \ldots, n$. Skupovi PRIMIJENI različitih nizova β_i nemaju zajedničkih znakova. Onda se prethodnih n produkcija zamjenjuje sljedećim $n + 1$ produkcijama: $A \to \alpha\, \langle\text{nastavak}\rangle$, $\langle\text{nastavak}\rangle \to \beta_\mathrm{i}$, i $= 1, 2, \ldots, n$.

Zadatak 73

Objasnite postupak uklanjanja lijeve rekurzije tijekom pretvorbe produkcija u $LL(1)$ oblik.

Produkcija je lijevo rekurzivna ako je nezavršni znak lijeve strane produkcije istodobno i na krajnje lijevom mjestu desne strane produkcije, $S \to S\, \alpha$. Neka je za nezavršni znak A zadano m (izravno) lijevo rekurzivnih produkcija $A \to A\, \alpha_\mathrm{i}$, i $= 1, 2, \ldots, n$ i n produkcija koje nisu izravno lijevo rekurzivne $A \to \beta_\mathrm{j}$, j $= 1, 2, \ldots, n$. Lijevo rekurzivne produkcije zamijene se produkcijama:
$A \to \beta_\mathrm{j}\, \langle Ponovi\rangle$ $\quad$ $\langle Ponovi\rangle \to \alpha_\mathrm{i}\, \langle Ponovi\rangle$ $\quad$ $\langle Ponovi\rangle \to \varepsilon$ $\quad$ $\langle Ponovi\rangle$ novi nezavršni znak

Zadatak 74
Navedite algoritam za izračunavanje ZAPOČINJE skupova za produkcije.

1. Računanje relacije *ZapočinjeIzravnoZnakom*:
   za znakove A i B vrijedi *ZapočinjeIzravnoZnakom*(A, B) ako i samo ako $A \to \alpha\, B\, \beta$, $\alpha \stackrel{*}{\Rightarrow} \varepsilon$

2. Računanje relacije *ZapočinjeZnakom*:
   za znakove A i B vrijedi *ZapočinjeZnakom*(A, B) ako i samo ako je iz znaka A moguće generirati niz koji započinje znakom B; napomena: relacija *ZapočinjeZnakom* jest refleksivna

3. (a) Računanje skupova ZAPOČINJE za nezavršne znakove
   završni je znak b u skupu ZAPOČINJE(X) $\iff$ vrijedi *ZapočinjeZnakom*(X, b)

   (b) Računanje skupova ZAPOČINJE za produkcije
   ZAPOČINJE$(X \to Z_1 Z_2 \ldots Z_n Y \alpha)$
   $=$ ZAPOČINJE$(Z_1) \cup$ ZAPOČINJE$(Z_2) \cup \cdots \cup$ ZAPOČINJE$(Z_n) \cup$ ZAPOČINJE(Y)

Zadatak 75
Opišite algoritam za izračunavanje relacije *Ispred*.

1. Računanje reacije *IzravnoIspredZnaka*:
   IzravnoIspredZnaka$(A,\, B)$ $\iff$ $D \to \alpha\, A\, \beta\, B\, \gamma$, $\beta \stackrel{*}{\Rightarrow} \varepsilon$

2. Računanje relacije *IzravniKraj*:
   IzravniKraj$(B,\, A)$ $\iff$ $A \to \alpha\, B\, \beta$, $\beta \stackrel{*}{\Rightarrow} \varepsilon$

3. Računanje relacije *Kraj*:
   Kraj$(A,\, B)$ $\iff$ iz znaka A moguće generirati niz koji završava znakom B

4. Računanje relacije *Ispred*:
   Ispred$(A,\, B)$ $\iff$ *Kraj*$(A,\, X) \wedge$ *IzravnoIspredZnaka*$(X,\, Y) \wedge$ *ZapočinjeZnakom*$(Y,\, B)$

Zadatak 76
Navedite korake u računanju PRIMIJENI skupova za produkcije.

Skupovi PRIMIJENI računaju se primjenom izračunatih skupova ZAPOČINJE i SLIJEDI. Ako produkcija ne generira prazni niz ε, onda se PRIMIJENI određuje pomoću skupa ZAPOČINJE. Inače, skup PRIMIJENI računa se kao SLIJEDI$(\varepsilon) \cup$ ZAPOČINJE(lijeva strana produkcije).

Zadatak 77
Opišite postupak računanja skupova SLIJEDI za prazne nezavršne znakove.

Na temelju tablice *Ispred* moguće je izravno odrediti skupove SLIJEDI za sve prazne nezavršne znakove.

Zadatak 78
Navedite i kratko opišite podatkovnu strukturu sintaksnog analizatora.

Podatkovnu strukturu sintaksnog analizatora čine globalni i lokalni podaci. Globalnu strukturu sačinjavaju tablica znakova (KROS, identifikatora i uniformnih znakova) i stog (za gradnju sintaksnog stabla). Lokalna struktura podataka sintaksnog analizatora gradi se za potrebe parsiranja.

Zadatak 79
Opišite kako se izvodi nadziranje i oporavak od pogrešaka kod LR-parsiranja.

Postupak traženja sinkronizacijskog znaka prekida sintaksnu analizu programske cjeline u kojoj je pronađena pogreška. Postupak oporavka od pogreške uzima sa stoga dio stanja koji pripada analizi nezavršnog znaka koji zadaje cjelinu u kojoj je došlo do sintaksne pogreške. Kazaljka koja se koristi tijekom čitanja pomiče se na znak koji slijedi nakon znakova generiranih iz problematičnog nezavršnog znaka i u ulaznom se spremiku traži znak ulančavanja naredbi (npr. ;) ili oznaka kraja bloka naredbi (npr. }). U postupku lokalnih promjena svi elementi tablice LR parsera koji označavaju akciju $Odbaci()$ zasebno se analiziraju i na temelju pravila izvornog jezika odredi se najvjerojatnija pogreška korisnika. Na temelju pretpostavljene pogreške odrede se akcije promjene sadržaja stoga i ulaznog niza.

Zadatak 81
Navedite i definirajte korake algoritma izgradnje kanonskog $LR(1)$-parsera

Iz produkcija gramatike najprije se napravi ε-NKA pa zatim DKA. Retci tablica *Akcija* i *NovoStanje* označavaju se završnim znakovima; stupci *Akcija* označavaju se završnim znakovima i oznakom kraja niza $\perp$, a stupci *NovoStanje* nezavršnim znakovima gramatike.
Tablica *Akcija* označava se na sljedeći način:

1. ako je $LR(1)$ stavka $A \to \alpha \bullet a\beta$, $\{a_1, a_2, \ldots, a_n\}$ u stanju s i ako je $\delta(s, a) = t$ prijelaz DKA, onda se za završni znak gramatike a i stanje s definira $Akcija[s, a] = Pomakni(t)$

2. ako je $LR(1)$ stavka $A \to \alpha \bullet$, $\{a_1, a_2, \ldots, a_n\}$ u stanju s onda se za sve završne znakove a koji su iz skupa $\{a_1, a_2, \ldots, a_n\}$ i stanje s definira $Akcija[s, a] = Reduciraj(A \to \alpha)$

3. ako je $LR(1)$ stavka $S' \to S \bullet$, $\{\perp\}$ u stanju s onda se za oznaku kraja niza $\perp$ i stanje s definira $Akcija[s, \perp] = Prihvati()$

Tablica *NovoStanje* popunjava se na sljedeći način:

1. Ako je $\delta(s, A) = t$ prijelaz DKA, onda se za nezavršni znak A i stanje s definira $NovoStanje[s, A] = Stavi(\mathrm{t})$

Svi neoznačeni elementi tablice označavaju akciju $Odbaci()$. Početno stanje označeno je $LR(1)$ stavkom $S' \to \bullet S$, $\{\perp\}$.

Zadatak 82
Opišite postupak izgradnje potisnog automata za S-gramatiku.

$G = (V, T, P, S) \rightarrow M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$
potisni automat ima samo jedno stanje, $Q = \{q_0\}$
skup ulaznih znakova proširen je oznakom kraja niza, $\Sigma = T \cup \{\perp\}$
skup znakova potisnog automata proširen je oznakom dna stoga, $\Gamma = \{\nabla\} \cup V \cup T'$, T' - svi završni znakovi koji su na desnim stranama produkcija, ali ne isključivo na krajnje lijevim mjestima
na početku rada na stogu je oznaka dna stoga, ∇ i početni nezvaršni znak S
funkcija prijelaza definira se dvodimenzionalnom tablicom: retci su znakovi stoga, stupci ulazni znakovi
elementi (ćelije) tablice označavaju akciju PA na ulaznom nizu i akciju na stogu
tablica se popunjava na sljedeći način:

$A \rightarrow b\alpha$, α niz završnih i nezavršnih znakova gramatike $\Rightarrow$ *Zamijeni*(α^r); *Pomakni*;
$A \rightarrow b \Rightarrow$ *Izvuci*; *Pomakni*;
element tablice $(\nabla, \perp) \Rightarrow$ *Prihvati*;
svi ostali elementi tablice $\Rightarrow$ *Odbaci*;

Zadatak 83
Definirajte $LL(1)$-gramatiku i kratko opišite konstrukciju potisnog automata za $LL(1)$-gramatiku.

$LL(1)$-gramatika dobiva se proširenjem produkcija Q-gramatike produkcijama koje na desnoj strani na krajnje lijevom mjestu mogu imati nezavršne znakove. Tablica se popunjava na sljedeći način:

- vrijede pravila kao i kod S-gramatike za produkcije oblika $A \rightarrow b\alpha$ i $A \rightarrow b$
- $A \rightarrow \varepsilon \Rightarrow$ u redak tablice A i sve stupce određene znakovima iz skupa PRIMIJENI($A \rightarrow \varepsilon$) zapisuju se akcije *Izvuci*; *Zadrži*;
- $A \rightarrow \alpha$, niz α započinje nezavršnim znakom $\Rightarrow$ u redak tablice A i sve stupce određene znakovima skupa PRIMIJENI($A \rightarrow \alpha$) zapisuju se akcije *Zamijeni*(α^r); *Zadrži*;
- Ako je za nezavršni znak A zadana prazna produkcija i ako u prethodnim koracima nije popunjen neki od elemenata u retku A, onda se za taj element tablice definira akcija *Odbaci* ili akcije ε-produkcije *Izvuci*; *Zadrži*;
- za element tablice $[b, b]$, b završni znak gramatike, $b \in \Gamma \Rightarrow$ *Izvuci*; *Pomakni*;

Zadatak 84
Navedite i općenito objasnite (ne na primjeru) sve akcije potisnog automata konstruiranog na osnovi $LL(1)$ gramatike.

1. *Zamijeni*(α^r) - nezavršni znak koji je na vrhu stoga zamjenjuje se nizom α^r

2. *Izvuci* - uzima se znak koji je na vrhu stoga
3. *Pomakni* - glava za čitanje pomiče se na sljedeći znak ulaznog niza
4. *Zadrži* - glava za čitanje ne miče se na sljedeći znak ulaznog niza
5. *Prihvati* - ulazni niz se prihvaća
6. *Odbaci* - ulazni niz se ne prihvaća

Zadatak 85
Navedite zadatke sintaksnog analizatora.

1. Grupiranje uniformnih znakova u sintaksne cjeline (izrazi, naredbe, blokovi narebi, program)
2. Provjera sintaksnih pravila (struktura izraza, naredbi i cjelokupnog programa)
3. Stvara hijerarhiju sintaksnih cjelina (jednoznačno se određuje za sve sintaksne cjeline)
4. Određivanje mjesta i opis pogreške
5. Izvodi postupak oporavka od pogreške
6. Gradi sintaksno stablo

Zadatak 86
Općenito definirajte ulaze i izlaze iz programa generatora sintaksnog analizatora i programa sintaksnog analizatora ako je sintaksni analizator ostvaren kao zasebni prolaz jezičnog procesora.

Generator sintaksnog analizatora na ulaz prima opis procesa sintaksnog analizatora (nezavršne i završne znakove gramatike, sinkronizacijski završni znakovi i produkcije gramatike). Izlaz generatora bit će izvorni kod (u jeziku izgradnje) sintaksnog analizatora. Sintaksni analizator na ulazu dobiva niz leksičkih jedinki koje čita i stvara sintaksno stablo (koje je izlaz sintaksnog analizatora).

Zadatak 87
Objasnite pojmove parsiranje, parsiranje od dna prema vrhu i parsiranje od vrha prema dnu.

parsiranje → postupak prepoznavanja niza i gradnja generativnog stabla na temelju zadanih produkcija kontekstno neovisne gramatike
parsiranje od vrha prema dnu → gradnja generativnog stabla započinje vrhom koji je označen početnim nezavršnim znakom gramatike i nastavlja se prmjenom produkcija gramtike na čvorove generativnog stabla koji su označeni nezavršnim znakovima gramatike
parsiranje od dna prema vrhu → gradnja generativnog stabla započinje listovima koji su označeni završnim znakovima gramatike i nastavlja se primjenom desnih strana produkcija gramatike na prethodno izgrađene čvorove

Zadatak 88
Neovisno poredajte gramatike $LL(1)$, S i Q te gramatike $LALR(1)$, $SLR(1)$, $LR(0)$ i $LR(1)$ uzlazno po općenitosti.

$S < Q < LL(1)$ gramatika
$LR(0) < SLR(1) < LALR(1) < LR(1)$ gramatika

Zadatak 89
Objasnite namjenu programa Yacc te dijelova ulazne datoteke za program Yacc.

Program Yacc primjer je generatora parsera, a koristi se i za rješavanje problema koje je moguće svesti na problem prihvaćanja kontekstno neovisnih jezika. Ulaznu datoteku čini opis sintaksnog analizatora i ona se sastoji od deklaracija, pravila prevođenja i pomoćne C procedure.

Zadatak 90
Ukratko objasnite postupak oporavka od pogreške kod LR-parsiranja koji se zasniva na traženju sinkronizacijskih znakova.

Postupak traženja sinkronizacijskog znaka prekida sintaksnu analizu programske cjeline u kojoj je pronađena pogreška. Na primjer, neka je parser pronašao pogrešku u sintaksnoj cjelini zadanoj nezavršnim znakom gramatike `<Naredba>`. Postupak oporavka od pogreške uzima sa stoga dio stanja koji pripada analizi nezavršnog znaka `<Naredba>`. Kazaljka koja se koristi tijekom čitanja pomiče se na znak koji slijedi nakon znakova generiranih iz problematičnog nezavršnog znaka i u ulaznom se spremiku traži znak ulančavanja naredbi (npr. ;) ili oznaka kraja bloka naredbi (npr. }). Nakon što se sa stoga uzmu stanja koja pripadaju analizi znaka `<Naredba>`, na stog se stavlja stanje S za koje je definirana akcija prijelaza u neko drugo stanje primjenom nezavršnog znaka `<Naredba>`. Na stog se stavlja stanje *NovoStanje*[S, `<Naredba>`].

Zadatak 91
Navedite pet različitih vrsta sustava oznaka za opis sintaksnih pravila.

1. BNF sustav oznaka
2. COBOL sustav oznaka
3. sintaksna analiza primjenom Co-No tablica
4. kontekstno neovisna gramatika
5. regularni izrazi

Zadatak 92
Opišite postupak sintaksne analize zasnovane na tablici Co-No.

Ispravnost izvornog programa i prevođenje u strojni program zasniva se na poznavanju dva podatka: desnog i lijevog operatora. Postupak analize izvornog programa i generiranja ciljnog programa zadaje se dvodimenzionalnom tablicom veličine $N \times N$, N broj operatora koji se koriste u izvornom programu.

Ako je par operatora dozvoljen sintaksnim pravilima, element tablice određen tim parom operatora označava jednu od akcija generatora ciljnog programa. U protivnom se u element tablice upisuje greška. (U naredbama je moguće zadati aritmetičke izraze koji imaju najviše dva operanda; koriste se operacije zbrajanja, oduzimanja, množenja i dijeljenja; znak pridruživanja je $\rightarrow$).

Zadatak 93
Objasnite parsiranje od dna prema vrhu tehnikom *Pomakni-Reduciraj*. Opišite tablice koje se koriste u parsiranju.

Na temelju kôda samo jednog znaka na vrhu stoga moguće je odrediti koji je niz znakova na vrhu stoga pa je moguće primjeniti akciju *Reduciraj* bez čitanja znakova na vrhu stoga i usporedbe sa znakovima desnih strana produkcija.
tablica *Pomakni/Reduciraj* $\rightarrow$ na temelju kôdiranog znaka na vrhu stoga i pročitanog znaka ulaznog niza odlučuje o primjeni akcije *Pomakni* ili *Reduciraj*
tablica *Stavi* $\rightarrow$ određuje kôd znaka koji se stavlja na vrh stoga

Zadatak 94
Opišite algoritme na kojima se zasnivaju postupci oporavka od pogreške kod sintaksne analize.

Algoritam odbacivanja znakova ulaznog niza sve dok se ne prepozna jedan od sinkronizacijskih znakova $\rightarrow$ sinkronizacijski znakovi su znakovi skupa SLIJEDI(A), ZAPOČINJE(A) ili definiraju sintaksne cjeline (A nezavršni znak na vrhu stoga). Ako odbacivanjem znakova ulaznog niza pročita znak iz skupa SLIJEDI(A) PA odbacuje A s vrha stoga i nastavlja parsiranje ulaznog niza; ako pročita znak iz skupa ZAPOČINJE(A) ostavlja A na vrhu stoga. Za ostale algoritme vidi zadatake 79. i 90.

Zadatak 95
Objasnite parsiranje od dna prema vrhu metodom prednosti operatora, relaciju prednosti, akcije parsera i određivanje uzorka za zamjenu.

Relacija prednosti određuje se za završne znakove a i b. Tablica u kojoj su određene relacije prednosti koristi se tijekom parsiranja niza. Akcije parsera su *Pomakni* (ako je prednost završnog znaka na vrhu stoga manja ili jednaka od prednosti znaka u ulaznom nizu) i *Reduciraj* (prednost završnog znaka na vrhu stoga veća je od prednosti znaka u ulaznom nizu). Uzorak za zamjenu čine svi znakovi sa stoga koji imaju manju prednost od prethodno uzetog znaka.

Zadatak 96
Objasnite razlike u ostvarenju parsera $LR(0)$, $SLR(1)$, $LALR$ i $LR(1)$ te navedite njihove prednosti i nedostatke.

- $LR(0)$ $\rightarrow$ parsiranje od dna prema vrhu, čita još 0 znakova desne strane prije primjene redukcije
- $SLR(1)$ $\rightarrow$ prednost jednostavnost, nedostatak nemogućnost primjene na velik skup jezika
- $LALR$ $\rightarrow$ nije programski zahtjevan kao LR postupak, obuhvaća manji skup jezika od kanonskog LR postupka

- $LR(1) \rightarrow$ čita još 1 znak ulaznog niza prije primjene redukcije

Zadatak 97
Objasnite konstrukciju ε-NKA u postupku izgradnje $SLR(1)$-parsera.

1. stanja su sve LR stavke gramatike i početno stanje q_0
2. skup ulaznih znakova unija je skupa nezavršnih i završnih znakova gramatike, $\Sigma = T \cup V$
3. sva stanja su u skupu prihvatljivih stanja, $F = Q$
4. funkcija prijelaza δ definira se kao:
   - $\delta(q_0, \varepsilon) = \{S \rightarrow \bullet\alpha \mid S \rightarrow \alpha$ produkcija gramatike$\}$
   - $\delta((A \rightarrow \alpha\bullet X\beta,\ X) = \{A \rightarrow \alpha X\bullet\beta\}$, ako automat pročita znak X, točka se pomiče iza X, (X završni ili nezavršni znak gramatike)
   - $\delta(A \rightarrow \alpha\bullet X\beta,\ \varepsilon) = \{B \rightarrow \bullet\gamma, \mid B \rightarrow \gamma$ produkcija gramatike$\}$, ako je nezavršni znak B iza točke, onda ε-prijelaz pokreće analizu tog nezavršnog znaka

Zadatak 98
Navedite i općenito objasnite sve četiri akcije potisnog automata konstruiranog na osnovi $LR(1)$ gramatike.

1. *Pomakni*(t) - stavlja stanje t na vrh stoga i pomiče glavu za čitanje na sljedeći znak ulaznog niza
2. *Reduciraj*$(A \rightarrow \alpha)$ - primjenjuje se redukcija (znakovi na vrhu stoga zamjenjuju se nezavršnim znakom)
3. *Stavi*(t) - na stog stavlja znak t na temelju tablice *NovoStanje*
4. *Prihvati*() - ulazni se niz prihvaća

Zadatak 99
Navedite zahtjeve koje mora ispuniti detekcija pogrešaka u sintaksnom analizatoru.

Zahtjeva se da postupak određivanja mjesta i opisa pogrešaka precizno odredi mjesto pogreške, kratko i jasno opiše pogrešku i da značajno ne uspori rad sintaksnog analizatora.

Zadatak 100
Objasnite sustav oznaka COBOL.

mala slova = varijabla, velika slova = konstanta koju je moguće izostaviti, podcrtana velika slova = konstanta koju nije moguće izostaviti, znak do znaka = nadovezivanje, [ ] = neobavezni izbor jedne od zadanih mogućnosti, { } = obavezni izbor jedne od zadanih mogćnosti, ... = ponavljanje prethodne sintaksne cjeline; zapis pravila da je potrebno odabrati (ili $*$ i zatim proizvoljan broj znakova iz skupa $\{A, B, C, D,)\}$ u sustavu oznaka COBOL:

$$\left\{ \begin{array}{c} (\\ * \end{array} \right\} \quad \left[\left\{ \begin{array}{c} A \\ B \\ C \\ D \\ ) \end{array} \right\} \right] \quad \ldots$$

Zadatak 101
Opišite postupak oporavka od pogrešaka u sintaksnom analizatoru.

Nakon što pronađe prvu pogrešku, sintaksni analizator nastavlja daljnju analizu u cilju pronalaženja ostalih pogrešaka. Tijekom postupka oporavka od pogreške sintaksni analizator promijeni stanje tako da omogući daljnji rad sintaksnog analizatora. Postupci oporavka od pogreške zasnivaju se na sljedećim algoritmima: traženje sinkronizacijskog znaka (izbacuje sve uniformne znakove do prvog sinkronizacijskog znaka), algoritam lokalnih promjena (izbacuje, zamjenjuje ili dodaje uniformne znakove da dobije prefiks neanaliziranog dijela programa koji zadovoljava sintaksna pravila), dodatne produkcije koje uključuju pogreške i algoritmi globalnih promjena (minimiziraju broj promjena u cjelokupnom izvornom programu potrebnih da se postigne sintaksno ispravan izvorni program).

Zadatak 102
Objasnite akcije parsera od dna prema vrhu koji koristi tehniku *Pomakni-Pronađi*. Opišite proturječja koja se pojavljuju.

Pomakni $\rightarrow$ pročitani znak stavlja se na vrh stoga, kazaljka za čitanje znakova miče se jedan znak desno
Reduciraj $\rightarrow$ ako je na vrhu stoga uzorak zamjenu s vrha stoga uzimaju se znakovi desne strane produkcije i na vrh se stavi nezavršni znak lijeve strane produkcije
Prihvati/*Odbaci* $\rightarrow$ prihvaćanje/ne prihvaćanje ulaznog niza
Dvije osnovne vrste proturječja su *Pomakni*/*Reduciraj* i *Reduciraj*/*Reduciraj*.

Zadatak 103

Uklonite lijevu rekurziju iz sljedeće gramatike.

<S> → a<A>b<B>a | b<B>a<A>b

<A> → <A>a<B>b | <B>a | a

<B> → <A>b | b

1. korak:

<S> → a<A>b<B>a | b<B>a<A>b

<A> → <A>a<B>b | <A>ba | ba | a

<B> → <A>a<B>bb | <B>ab | ab | b

2. korak:

<S> → a<A>b<B>a | b<B>a<A>b

<A> → ba<C> | a<C>

<C> → a<B>b<C> | ba<C> | ε

<B> → <A>a<B>bb | ab<D> | b<D>

<D> → ab<D> | ε

Potrebno još dodatno srediti da bi gramatika bila npr. *LL(1)*.

Zadatak 104

Uklonite lijevu rekurziju iz dane gramatike.

<S> → a<A>b<B>a | b<B>a<A>b

<A> → <B>b | b

<B> → <A>a | <B>a<A>b | a

1. korak:

<S> → a<A>b<B>a | b<B>a<A>b

<A> → <A>ab | <B>a<A>bb | ab | b

<B> → <B>ba | ba | <B>a<A>b | a

2. korak:

<S> → a<A>b<B>a | b<B>a<A>b

<A> → <B>a<A>bb<C> | ab<C> | b<C>

<C> → ab<C> | ε

<B> → ba<D> | a<D>

<D> → ba<D> | a<A>b<D> | ε

Zadatak 109

Za zadanu Q-gramatiku konstruirajte potisni automat. Tijekom parsiranja ulaznog niza na vrhu stoga redom se pojavljuju sljedeći znakovi: <S>, <A>, <B>, <B>, ∇. Koji su se ulazni znakovi sigurno nalazili u parsiranom nizu?

<S> $\rightarrow$ a<A><B> | b<A><B>

<A> $\rightarrow$ a<A> | b<A> | c | ε

<B> $\rightarrow$ d | e<B> | f<A> | ε

	a	b	c	d	e	f	$\perp$
<S>	(1)	(1)	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*
<A>	(2)	(2)	(3)	(5)	(5)	(5)	(5)
<B>	*Odbaci*	*Odbaci*	*Odbaci*	(3)	(4)	(2)	(5)
∇	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Prihvati*

(1) *Zamijeni*(<B><A>); *Pomakni*;

(2) *Zamijeni*(<A>); *Pomakni*;

(3) *Izvuci*; *Pomakni*;

(4) *Zamijeni*(<B>); *Pomakni*;

(5) *Izvuci*; *Zadrži*;

Ako imamo produkciju $A \rightarrow \varepsilon$, onda se u redak tablice A i sve stupce određene znakovima skupa PRIMIJENI($A \rightarrow \varepsilon$) = SLIJEDI($A$) zapisuju akcije *Izvuci*; *Zadrži*;

čitamo	akcija PA	stog
-	-	∇<S>
a ili b	1	∇<B><A>
c	3	∇<B>
e	4	∇<B>
d	3	∇

Zadatak 110

Napišite gramatiku na temelju koje je konstruiran navedeni potisni automat te odredite o kojoj se gramatici radi.

	a	b	$\perp$
S	1	2	8
A	3	4	8
B	5	6	8
a	7	8	8
b	8	7	8
∇	8	8	9

1: *Pomakni*; *Zamijeni*(aBbA);
2: *Pomakni*; *Zamijeni*(bAaB);
3: *Pomakni*; *Zamijeni*(bA);
4: *Zadrži*; *Zamijeni*(B);
5: *Zadrži*; *Izvuci*;
6: *Pomakni*; *Zamijeni*(aB);
7: *Pomakni*; *Izvuci*;
8: *Odbaci*;
9: *Prihvati*;

S $\rightarrow$ aAbBa | bBaAb

A $\rightarrow$ aAb | B (zaključak da je b u skupu PRIMIJENI($A \rightarrow B$) = ZAPOCINJE(B))

B $\rightarrow$ bBa | ε (zaključak da je a u skupu PRIMIJENI($B \rightarrow \varepsilon$))

Zadatak 111

Za zadanu gramatiku konstruirajte konačni potisni automat i izrazite ga pomoću tablice. Prikažite rad potisnog automata na nizu `ebabc`.

`<S>` $\rightarrow$ `<A><B> | d<C>`

`<A>` $\rightarrow$ `a<C><B>` | ε

`<B>` $\rightarrow$ `b<C> | e<S>`

`<C>` $\rightarrow$ `a<B> | c`

	a	b	c	d	e	$\perp$
<S>	(1)	(1)	*Odbaci*	(2)	(1)	*Odbaci*
<A>	(3)	(4)	*Odbaci*	*Odbaci*	(4)	*Odbaci*
<B>	*Odbaci*	(2)	*Odbaci*	*Odbaci*	(5)	*Odbaci*
<C>	(6)	*Odbaci*	(7)	*Odbaci*	*Odbaci*	*Odbaci*
∇	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Prihvati*

(1) *Zamijeni*(<B><A>); *Zadrži*;

(2) *Zamijeni*(<C>); *Pomakni*;

(3) *Zamijeni*(<C><B>); *Pomakni*;

(4) *Izvuci*; *Zadrži*;

(5) *Zamijeni*(<S>); *Pomakni*;

(6) *Zamijeni*(<B>); *Pomakni*;

(7) *Izvuci*; *Pomakni*;

primjena na ulazni niz `ebabc`$\perp$:

čitamo	akcija PA	stog
-	-	∇S
e	1	∇BA
b	4	∇B
b	2	∇C
a	6	∇B
b	2	∇C
c	7	∇
$\perp$	*Prihvati*	

Zadatak 112

Iz navedenog potisnog automata rekonstruirajte gramatiku.

	a	b	c	d	e	⊥
<S>	1	9	9	1	1	9
<A>	6	9	9	9	7	9
<B>	2	9	9	3	2	9
<C>	4	5	9	9	9	9
∇	9	9	9	9	9	8

1: *Zadrži*; *Zamijeni*(d<A>b<B>);

2: *Zadrži*; *Zamijeni*(d<A>);

3: *Pomakni*; *Zamijeni*(<C>c<S>);

4: *Pomakni*; *Zamijeni*(<C>);

5: *Zadrži*; *Izvuci*;

6: *Pomakni*; *Zamijeni*(<A>);

7: *Pomakni*; *Izvuci*;

8: *Prihvati*;

9: *Odbaci*;

<S> → <B>b<A>d (zaključak da su a, d i e u skupu PRIMIJENI($S \to BbAd$) = ZAPOCINJE(B))

<A> → a<A> | e

<B> → <A>d | d<S>c<C> (zaključak da su a i e u skupu PRIMIJENI($B \to Ad$) = ZAPOCINJE(A))

<C> → a<C> | ε (zaključak da je b u skupu PRIMIJENI($C \to \varepsilon$) = SLIJEDI(C), dokaz je u prvoj produkciji)

Zadatak 113

Za zadanu gramatiku konstruirajte konačni potisni automat i izrazite ga pomoću tablice. Prikažite rad potisnog automata na nizu `cdabed`.

`<S>` $\rightarrow$ `a<A>bc<B> | c<A>`

`<A>` $\rightarrow$ `a<A> | b | d<A>e<C>`

`<B>` $\rightarrow$ `b<B> | e<C>`

`<C>` $\rightarrow$ `c | d | e`

	a	b	c	d	e	$\perp$
<S>	(1)	*Odbaci*	(2)	*Odbaci*	*Odbaci*	*Odbaci*
<A>	(2)	(3)	*Odbaci*	(4)	*Odbaci*	*Odbaci*
<B>	*Odbaci*	(5)	*Odbaci*	*Odbaci*	(6)	*Odbaci*
<C>	*Odbaci*	*Odbaci*	(3)	(3)	(3)	*Odbaci*
b	*Odbaci*	(3)	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*
c	*Odbaci*	*Odbaci*	(3)	*Odbaci*	*Odbaci*	*Odbaci*
e	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	(3)	*Odbaci*
∇	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Odbaci*	*Prihvati*

(1) *Zamijeni*(`<B>cb<A>`); *Pomakni*;

(2) *Zamijeni*(`<A>`); *Pomakni*;

(3) *Izvuci*; *Pomakni*;

(4) *Zamijeni*(`<C>e<A>`); *Pomakni*;

(5) *Zamijeni*(`<B>`); *Pomakni*;

(6) *Zamijeni*(`<C>`); *Pomakni*;

primjena na ulazni niz `cdabed`$\perp$:

čitamo	akcija PA	stog
-	-	∇S
c	2	∇A
d	4	∇CeA
a	2	∇CeA
b	3	∇Ce
e	3	∇C
d	3	∇
$\perp$	*Prihvati*	

Zadatak 127

Zadana je Co-No tablica. Odredite rezultat izvodenja zadanog programa.

`,7→x,12→y,x*y→x,3→z,x/z→y,x-y/2→x,x*2/y+z→x,x*6-y+z*2-9*x→y,x+y/2→z,`

	,	→	∗	/	+	−
,	greška	dohvati	dohvati	dohvati	dohvati	dohvati
→	spremi	greška	greška	greška	greška	greška
∗	greška	pomnoži	pomnoži	pomnoži	pomnoži	pomnoži
/	greška	podijeli	podijeli	podijeli	podijeli	podijeli
+	greška	zbroji	zbroji	zbroji	zbroji	zbroji
−	greška	oduzmi	oduzmi	oduzmi	oduzmi	oduzmi

dohvati	7
spremi	$x = 7$
dohvati	12
spremi	$y = 12$
dohvati	x
pomnoži	$7 \cdot 12$
spremi	$x = 84$
dohvati	3
spremi	$z = 3$
dohvati	x
podijeli	$84 : 3$
spremi	$y = 28$

dohvati	x
oduzmi	$x - y$
podijeli	$56 : 2$
spremi	$x = 28$
dohvati	x
pomnoži	$28 \cdot 2$
podijeli	$56 : 28$
zbroji	$2 + 3$
spremi	$x = 5$
dohvati	x
pomnoži	$5 \cdot 6$
oduzmi	$30 - 28$

zbroji	$2 + 3$
pomnoži	$5 \cdot 2$
oduzmi	$10 - 9$
pomnoži	$1 \cdot 5$
spremi	$y = 5$
dohvati	x
zbroji	$5 + 5$
podijeli	$10 : 2$
spremi	$z = 5$

Zadatak 134

Na temelju zadane operatorske gramatike izgradite tablicu relacija prednosti. Prikažite parsiranje niza $(a \vee a \wedge \neg a) \wedge \neg a \vee a$ pomoću izgrađene tablice.

<E> → <E> ∨ <T> | <T>

<T> → <T> ∧ <P> | <P>

<P> → <N> | ¬<N>

<N> → (<E>)

<N> → a

	$\vee$	$\wedge$	$\neg$	(	)	a	$\perp$
$\vee$	$\Rightarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$
$\wedge$	$\Rightarrow$	$\Rightarrow$	$\Leftarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$
$\neg$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$
(	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftrightarrow$	$\Leftarrow$	
)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$		$\Rightarrow$		$\Rightarrow$
a	$\Rightarrow$	$\Rightarrow$	$\Leftarrow$		$\Rightarrow$		$\Rightarrow$
∇	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$		$\Leftarrow$	

stog	odluka	ulazni niz	akcija
∇	$\Leftarrow$	$(a \vee a \wedge \neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow ($	$\Leftarrow$	$a \vee a \wedge \neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow a$	$\Rightarrow$	$\vee a \wedge \neg a) \wedge \neg a \vee a \perp$	*Reduciraj*()
$\nabla \Leftarrow ($	$\Leftarrow$	$\vee a \wedge \neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow \vee$	$\Leftarrow$	$a \wedge \neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow \vee \Leftarrow a$	$\Rightarrow$	$\wedge \neg a) \wedge \neg a \vee a \perp$	*Reduciraj*()
$\nabla \Leftarrow (\Leftarrow \vee$	$\Leftarrow$	$\wedge \neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow \vee \Leftarrow \wedge$	$\Leftarrow$	$\neg a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow \vee \Leftarrow \wedge \Leftarrow \neg$	$\Leftarrow$	$a) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftarrow \vee \Leftarrow \wedge \Leftarrow \neg \Leftarrow a$	$\Rightarrow$	$) \wedge \neg a \vee a \perp$	*Reduciraj*()
$\nabla \Leftarrow ($	$\Leftrightarrow$	$) \wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow (\Leftrightarrow)$	$\Rightarrow$	$\wedge \neg a \vee a \perp$	*Reduciraj*()
∇	$\Leftarrow$	$\wedge \neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow \wedge$	$\Leftarrow$	$\neg a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow \wedge \Leftarrow \neg$	$\Leftarrow$	$a \vee a \perp$	*Pomakni*()
$\nabla \Leftarrow \wedge \Leftarrow \neg \Leftarrow a$	$\Rightarrow$	$\vee a \perp$	*Reduciraj*()
$\nabla \Leftarrow \wedge \Leftarrow \neg$	$\Leftarrow$	$\vee a \perp$	*Pomakni*()
$\nabla \Leftarrow \wedge \Leftarrow \neg \Leftarrow \vee$	$\Leftarrow$	$a \perp$	*Pomakni*()
$\nabla \Leftarrow \wedge \Leftarrow \neg \Leftarrow \vee \Leftarrow a$	$\Rightarrow$	$\perp$	*Reduciraj*()
∇	$\Rightarrow$	$\perp$	*Prihvati*()

Zadatak 136

Odredite PRIMIJENI skupove svih produkcija u zadanoj gramatici. Odredite je li gramatika $LL(1)$ i obrazložite.

<S> → a<S>b<S> | c<D><A> | ε

<A> → <B>c | ba<B>

<B> → a<D> | de<C> | e<C>

<C> → <D>f | a<B>c

<D> → e<D> | ε

PRIMIJENI(<S> → a<S>b<S>) = ZAPOČINJE(a<S>b<S>) = {a}

PRIMIJENI(<S> → c<D><A>) = ZAPOČINJE(c<D><A>) = {c}

PRIMIJENI(<S> → ε) = ZAPOČINJE(ε) ∪ SLIJEDI(<S>) = {b, ⊥}

PRIMIJENI(<A> → <B>c) = ZAPOČINJE(<B>c) = {a, d, e}

PRIMIJENI(<A> → ba<B>) = ZAPOČINJE(ba<B>) = {b}

PRIMIJENI(<B> → a<D>) = ZAPOČINJE(a<D>) = {a}

PRIMIJENI(<B> → de<C>) = ZAPOČINJE(de<C>) = {d}

PRIMIJENI(<B> → e<C>) = ZAPOČINJE(e<C>) = {e}

PRIMIJENI(<C> → <D>f) = ZAPOČINJE(<D>f) = {a, b, d, e, f, ⊥}

PRIMIJENI(<C> → a<B>c) = ZAPOČINJE(a<B>c) = {a}

PRIMIJENI(<D> → e<D>) = ZAPOČINJE(e<D>) = {e}

PRIMIJENI(<D> → ε) =ZAPOČINJE(ε) ∪ SLIJEDI(<D>) = {a, b, d, e, f, ⊥}

Gramatika nije $LL(1)$ jer postoje produkcije koje imaju istu lijevu stranu i njihovi PRIMIJENI skupovi imaju zajedničkih elemenata.

Zadatak 152

Za zadanu gramatiku izgradite parser zasnovan na tehnici parsiranja *Pomakni-Pronađi*.

(1) <S> → b<A>

(2) <S> → p<A>m<C>

(3) <A> → d<S>a

(4) <A> → e

(5) <C> → d<A>

	a	b	d	e	p	m	⊥
<S>	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*1()
<A>	*Pronađi*2()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Pronađi*2()
<C>	*Pronađi*3()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*3()
a	*Pronađi*4()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*4()	*Pronađi*4()
b	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()
d	*Odbaci*()	*Pomakni*()	*Pomakni*()	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()
e	*Pronađi*5()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*5()	*Pronađi*5()
p	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()
m	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()
∇	*Odbaci*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Odbaci*()	*Odbaci*()

SLIJEDI(<S>) = {a, ⊥}

SLIJEDI(<A>) = {a, m, ⊥}

SLIJEDI(<C>) = {a, ⊥}

SLIJEDI(a) = {m, ⊥}

SLIJEDI(b) = {d, e}

SLIJEDI(d) = {b, d, e, p}

SLIJEDI(e) = {a, m, ⊥}

SLIJEDI(p) = {d, e}

SLIJEDI(m) = {d}

imamo relacije: $ReduciranZnakom(A, a)$, $ReduciranZnakom(A, \perp)$, $ReduciranZnakom(C, a)$, $ReduciranZnakom(C, \perp)$, $ReduciranZnakom(a, a)$, $ReduciranZnakom(a, m)$, $ReduciranZnakom(a, \perp)$, $ReduciranZnakom(e, a)$, $ReduciranZnakom(e, m)$, $ReduciranZnakom(e, \perp)$
ne definiramo relacije *ReduciranZnakom* za znakove koji nisu na krajnje desnom mjestu desnih strana produkcija

imamo relacije: $IspodZnaka(S, a)$, $IspodZnaka(A, m)$, $IspodZnaka(b, d)$, $IspodZnaka(b, e)$, $IspodZnaka(d, b)$, $IspodZnaka(d, d)$, $IspodZnaka(d, e)$, $IspodZnaka(d, p)$, $IspodZnaka(p, d)$, $IspodZnaka(p, e)$, $IspodZnaka(m, d)$, $IspodZnaka(\nabla, b)$, $IspodZnaka(\nabla, p)$
ne definiramo relacije *IspodZnaka* za znakove koji se nalaze isključivo na krajnje desnim mjestima desnih strana produkcija

```
Pronadi1(){
    ako Stog == nabla<S>
        Prihvati();
    inace
        Odbaci();
}

Pronadi2(){
    ako VrhStoga == b<A>
        Reduciraj1();
    inace ako VrhStoga == d<A>
        Reduciraj5();
    inace
        Odbaci();
}

Pronadi3(){
    ako VrhStoga == p<A>m<C>
        Reduciraj2();
    inace
        Odbaci();
}
```

```
Pronadi4(){
    ako VrhStoga == d<S>a
        Reduciraj3();
    inace
        Odbaci();
}

Pronadi5(){
    Reduciraj4();
}

Pomakni(){
    StaviNaStog(Procitani znak);
    Pomakni Kazalju Na Sljedeci Znak;
}
```

Zadatak 156

Za zadanu gramatiku konstruirajte parser zasnovan na tehnici *Pomakni-Pronađi*. Proturječja *Pomakni/Pronađi* razriješite u korist akcije *Pomakni*.

(1) <S> → aba<S>q<W>

(2) <S> → baf<Y>

(3) <W> → cc<S>

(4) <W> → q

(5) <Y> → caba<W>

(6) <Y> → ffa<S>q<W>

	a	b	c	f	q	⊥
<S>	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Pronađi*()
<W>	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*()	*Pronađi*()
<Y>	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Pronađi*()	*Pronađi*()
a	*Pomakni*()	*Pomakni*()	*Pomakni*()	*Pomakni*()	*Pomakni*()	*Odbaci*()
b	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()
c	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()
f	*Pomakni*()	*Odbaci*()	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()
q	*Odbaci*()	*Odbaci*()	*Pomakni*()	*Odbaci*()	*Pronađi*()	*Pronađi*()
∇	*Pomakni*()	*Pomakni*()	*Odbaci*()	*Odbaci*()	*Odbaci*()	*Odbaci*()

SLIJEDI(<S>) = {q, ⊥}

SLIJEDI(<W>) = {q, ⊥}

SLIJEDI(<Y>) = {q, ⊥}

imamo relacije: (pretvaramo ih u akcije *Pronađi*)

$ReduciranZnakom(S, \perp)$, $ReduciranZnakom(S, q)$, $ReduciranZnakom(W, q)$, $ReduciranZnakom(W, \perp)$, $ReduciranZnakom(Y, q)$, $ReduciranZnakom(Y, \perp)$, $ReduciranZnakom(q, q)$, $ReduciranZnakom(q, \perp)$

imamo relacije: (pretvaramo ih u akcije *Pomakni*)

$IspodZnaka(\nabla, a)$, $IspodZnaka(\nabla, b)$, $IspodZnaka(S, q)$, $IspodZnaka(a, a)$, $IspodZnaka(a, b)$, $IspodZnaka(a, c)$, $IspodZnaka(a, f)$, $IspodZnaka(a, q)$, $IspodZnaka(b, a)$, $IspodZnaka(c, a)$, $IspodZnaka(c, b)$, $IspodZnaka(f, a)$, $IspodZnaka(f, c)$, $IspodZnaka(f, f)$, $IspodZnaka(q, c)$, $IspodZnaka(q, q)$

```
Pronadi(){
    ako Stog == aba<S>q<W>
        Reduciraj1();
    inace ako Stog == baf<Y>
        Reduciraj2();
    inace ako Stog == cc<S>
        Reduciraj3();
    inace ako Stog == q
        Reduciraj4();
    inace ako Stog == caba<W>
        Reduciraj5();
    inace ako Stog == ffa<S>q<W>
        Reduciraj6();
    inace ako Stog == nabla <S> && Ulaz = kraj niza
        Prihvati();
    inace
        Odbaci();
}
```

stog	znak	ulazni niz	akcija
∇		$ababafcabaqqq\perp$	
∇	a	$babafcabaqqq\perp$	$Pomakni()$
∇a	b	$abafcabaqqq\perp$	$Pomakni()$
∇ab	a	$bafcabaqqq\perp$	$Pomakni()$
∇aba	b	$afcabaqqq\perp$	$Pomakni()$
$\nabla abab$	a	$fcabaqqq\perp$	$Pomakni()$
$\nabla ababa$	f	$cabaqqq\perp$	$Pomakni()$
$\nabla ababaf$	c	$abaqqq\perp$	$Pomakni()$
$\nabla ababafc$	a	$baqqq\perp$	$Pomakni()$
$\nabla ababafca$	b	$aqqq\perp$	$Pomakni()$
$\nabla ababafcab$	a	$qqq\perp$	$Pomakni()$
$\nabla ababafcaba$	q	$qq\perp$	$Pomakni()$
$\nabla ababafcabaq$	q	$q\perp$	$Pronađi()$
$\nabla ababafcabaW$	q	$q\perp$	$Pronađi()$
$\nabla ababafY$	q	$q\perp$	$Pronađi()$
$\nabla abaS$	q	$q\perp$	$Pomakni()$
$\nabla abaSq$	q	$\perp$	$Pomakni()$
$\nabla abaSqq$	q	$\perp$	$Pronađi()$
$\nabla abaSqW$	q	$\perp$	???

Zadatak 173

Prikažite korake tijekom parsiranja niza *bbbcb* primjenom zadanog $LR(1)$-parsera.

stanje	akcija			novo stanje		
	b	c	$\perp$	S	A	B
0	P1					
1	P2				S3	
2	P5					S6
3	P4					S7
4			R3			
5		R3				
6		P8				
7			Prihvati			
8	R2					

R1 = $Reduciraj(S \to bAB)$

R2 = $Reduciraj(A \to bBc)$

R3 = $Reduciraj(B \to b)$

stog	znak koji čitamo	akcija
$\nabla 0$	b	P1
$\nabla 0b1$	b	P2
$\nabla 0b1b2$	b	P5
$\nabla 0b1b2b5$	c	R3
$\nabla 0b1b2B6$	c	P8
$\nabla 0b1b2B6c8$	b	R2
$\nabla 0b1A3$	b	R2
$\nabla 0b1A3$	b	P4
$\nabla 0b1A3b4$	$\perp$	R3
$\nabla 0b1A3B7$	$\perp$	Prihvati

Zadatak 182

Izgradite $SLR(1)$-parser za zadanu gramatiku.

(1) $S \to AaBb$

(2) $A \to a$

(3) $B \to cA$

(4) $B \to \varepsilon$

stanje	akcija				novo stanje		
	a	b	c	$\perp$	S	A	B
0	P1					S2	
1	R2	R2					
2	P3						
3		R4	P4				S5
4	P1					S6	
5		P7					
6		R3					
7				Prihvati			

Zadatak 186

Izgradite $LR(1)$-parser za zadanu gramatiku.

(1) $S \to bA$

(2) $A \to Sa$

(3) $A \to \varepsilon$

stanje	akcija			novo stanje		
	a	b	$\perp$	S	A	B
0		P1				
1		P4	R3	S3	S2	
2			Prihvati			
3	P5		R2			
4	R3	P4	R3	S7	S6	
5			R2			
6	R1					
7	P8					
8	R2		R2			

4 Semantička analiza

Zadatak 205
Opišite postupak izgradnje potisnog automata za prijevodnu gramatiku.

1. PA ima samo jedno stanje q_0 koje je ujedno i početno stanje
2. skup ulaznih znakova PA je skup završnih znakova i oznaka kraja niza, $\Sigma = T \cup \perp$
3. skup znakova stoga PA su oznaka dna stoga ∇, skup nezavršnih znakova V, skup završnih znakova koji su na desnim stranama produkcija, ali ne isključivo na krajnje lijevim mjestima, te izlazni završni znakovi (samo ako je na desnoj strani produkcije desno od krajnje lijevog nezavršnog ili ulaznog završnog znaka)
4. tablica PA popunjava se na sljedeći način:
   - za produkciju $A \to \xi\, b\, \phi\, \alpha$ (gdje su ξ, ϕ nizovi izlaznih završnih znakova, α niz nezavršnih i završnih znakova koji ne započinje izlaznim završnim znakom) u redak A i stupac b zapisuje se akcija *Izlaz*($\xi\, \phi$); *Zamijeni*(α'); *Pomakni*;
   - ako je izlazni završni znak $\{\xi\}$ znak stoga, onda elementi tablice koji su u retku $\{\xi\}$ određuju akcije *Izlaz*(ξ); *Izvuci*; *Zadrži*;
   - ako je u ε-produkciji zadan niz izlaznih završnih znakova ξ, onda se za produkciju oblika $A \to \xi$ u redak A i stupce određene znakovima skupa PRIMIJENI($A \to \varepsilon$ zapisuju akcije *Izlaz*(ξ); *Izvuci*; *Zadrži*;
   - za produkciju $A \to \xi\, \alpha$ (gdje je α niz znakova koji započinje nezavršnim znakom gramatike, a ξ niz izlaznih završnih znakova gramatike) onda se u redak A i stupce određene znakovima skupa PRIMIJENI($A \to \alpha$ zapisuju akcije *Izlaz*(ξ); *Zamijeni*(α'); *Zadrži*;

Zadatak 206
Objasnite što je provjera vrijednosti obilježja i opišite pojedine postupke za provjeru vrijednosti obilježja.

Provjera vrijednosti obilježja koristi atributnu prijevodnu gramatiku. Ako se tijekom provjere vrijednosti obilježja ustanovi pogreška, onda semantički analizator ispisuje poruku o pogrešci, pridruži obilježju vrijednost *Pogreška*, pokrene postupak oporavka od pogreške i nastavi analizu izvornog programa.
Provjera vrijednosti obilježja naredbi deklaracija Tijekom obrade naredbi deklaracija provjerava se njihova ispravnost. U tablicu identifikatora zapisuju se vrijednosti obilježja.
Provjera vrijednosti obilježja izraza Za npr. produkciju $E_V \to$ **INT** svojstvu V pridruži se vrijednost obilježja konstante (**IDN** označava int konstantu npr. 123). Ako je produkcijom definirana varijabla, onda se vrijednost svojstva V određuje na temelju vrijednosti obilježja koje je zapisano u tablici identifikatora tijekom deklaracije varijable.
Provjera vrijednosti obilježja ulančanih naredbi Naredbe se ulančavaju primjenom operatora *;* Obilježjima naredbi pridružuju se dvije vrijednosti *BezPogreške* i *Pogreška*.

Zadatak 207
Opište svojstva L-atributne prijevodne gramatike.

Atributna je prijevodna gramatika L-atributna ako i samo ako vrijedi:

1. Vrijednost *nasljednog svojstva* znaka desne strane produkcije računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije koji su lijevo od zadanog znaka.

2. Vrijednost *izvedenog svojstva* nezavršnog znaka lijeve strane produkcije računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije.

3. Vrijednost izvedenog svojstva izlaznog završnog znaka računa se na temelju nasljednih svojstava tog istog izlaznog završnog znaka.

Zadatak 208
Nabrojite i objasnite formalne modele semantičkog analizatora.

- Model zasnovan na *prevođenju izvornog jezika* u jezik koji ima definiranu semantiku

- *Izvođenje na apstraktnom stroju* zasniva se na skupu pravila primjenom koji se simulira izvođenje izvornog programa. Apstraktno se računalo definira stanjima. Izvođenje programa simulira se skupom funkcija koje mijenjaju stanje apstraktnog računala.

- Formalni model koji koristi *skup aksioma*. Aksiomi su logičke tvrdnje koje korisnik zadaje u različitim dijelovima izvornog programa, a njima izriče očekivani rezultat izvođenja tog dijela izvornog programa. Semantička ispravnost programa zasniva se na provjeri istovjetnih statičkih logičkih tvrdnji i rezultata dinamičkog izvođenja programa.

Zadatak 211
Objasnite sintaksom vođenu semantičku analizu.

Sintaksom upravljani jezični procesori zasnovani su na primjeni atributne prijevodne gramatike, a sintaksni je analizator središnji proces analize izvornog i sinteze ciljnog programa. Sintaksni analizator pokreće izvođenje semantičkih akcija u cilju provjere semantičkih pravila. Semantičke se akcije ugrađuju u produkcije formalne gramatike kao izlazni završni znakovi. Produkcijama gramatike pridružuju se pravila računanja vrijednosti svojstava. Prijevodna gramatika proširena svojstvima i pravilima računanja vrijednosti tih svojstava naziva se atributna prijevodna gramatika.

Zadatak 212
Definirajte atributnu prijevodnu gramatiku.

Atributna je gramatika proširena prijevodna gramatika. Znakovima gramatike dodjeljuje se konačni skup svojstava i pravila računanja tih svojstava. Svojstva se dijele na *nasljedna* (vrijednost nasljednog svojstva znaka desne strane produkcije određuje se na temelju vrijednosti svojstava ostalih znakova koji su na lijevoj ili desnoj strani produkcije) i *izvedena* (vrijednost izvedenog svojstva nezavršnog znaka lijeve strane produkcije računa se na temelju svojstava ostalih znakova koji su na lijevoj ili desnoj strani produkcije).

Zadatak 213
Opišite algoritam provjere jednakosti tipova obilježja temeljen na provjeri jednakosti strukture obilježja.

Za potrebe ispitivanja jednakosti dva obilježja koristi se postupak ujednačavanja. Ujednačavanje obilježja s i t postupak je utvrđivanja jednakosti dva obilježja na način da se varijable u oba obilježja zamjene odgovarajućim zajedničkim vrijednostima. Tijekom postupka ujednačavanja uspoređuju se čvorovi stabala strukture. Ako su čvorovi stabala označeni istim konsturktorom ili ako je jedan od čvorova označen varijablom onda se oni združe u jednu grupu.

Zadatak 214
Objasnite kako se obrađuju izvedena svojstva izlaznih završnih znakova koji se ne stavljaju na stog.

Izlazni završni znak $\{Rezultat\}$ pokreće izvođenje semantičke akcije koja ispisuje vrijednost aritmetičkog izraza. Izlaznom završnom znaku $\{Rezultat\}$ dodjeljuje se svojstvo sa značenjem rezultata aritmetičkog izraza generiranog iz nezavršnog znaka S. $\{Rezultat_r\}$ oznaka izlaznog završnog znaka $\{Rezultat\}$ kojemu je dodijeljeno svojstvo r. Vrijednost aritmetičkog izraza računa se korak po korak primjenom produkcija gramatike. Međurezultati računanja prenose se sintaksnim stablom od ulaznih završnih znakova do izlaznog završnog znaka $\{Rezultat\}$. Izračunata vrijednost aritmetičkog izraza pridruži se svojstvu r izlaznog završnog znaka $\{Rezultat\}$.

Zadatak 215
Navedite i objasnite algoritam ispitivanja jednakosti obilježja konstantnih vrijednosti.

```
Jednakost(s, t) {
    if ((s i t jednake jednostavne vrijednosti) || (s i t ista korisnicka imena))
        return True
    else if ((s == Polje(s1, s2) && (t == Polje(t1, t2)))
        return ((Jednakost(s1, t1)) & (Jednakost(s2, t2)))
    else if ((s == Kazaljka(s1)) && (t == Kazaljka(t1)))
        return (Jednakost(s1, t1))
    else if ((s == s1 -> s2) && (t == t1 -> t2))
        return ((Jednakost(s1, t1)) & (Jednakost(s2, t2)))
    else if ((s == s1 x s2) && (t == t1 x t2))
        return ((Jednakost(s1, t1)) & (Jednakost(s2, t2)))
    else return False
}
```

Zadatak 217
Opišite korake gradnje atributnog generativnog stabla. Definirajte potpuno atributno generativno stablo.

1. Primjenom produkcija prijevodne gramatike izgradi se generativno stablo za zadani niz ulaznih završnih znakova.
2. Svojstvima ulaznih završnih znakova gramatike pridruže se pročitane vrijednosti.
3. Nasljednim svojstvima početnog nezavršnog znaka pridruže se početne vrijednosti koje su definirane zajedno s produkcijama gramatike.
4. Pretražuje se generativno stablo. Traži se svojstvo koje nema izračunatu vrijednost. Izračuna se vrijednost izabranog svojstva na temelju ostalih vrijednosti. Opisani se postupak nastavlja traženjem sljedećeg svojstva i računanjem njegovih vrijednosti.

Atributno je generativno stablo potpuno ako je moguće izračunati vrijednost svih svojstava koja su dodijeljena svim znakovima gramatike.

Zadatak 218
Objasnite kako se obrađuju svojstva izvorišta koja nemaju dostupne vrijednosti.

Budući da vrijednosti svojstava izvorišta nisu dostupne, potisni automat stavlja na stog kazaljku koja pokazuje na listu polja dodijeljenih svojstvima odredišta. Vrijednosti kazaljki prenose se od vrha generativnog stabla prema dnu sve dok vrijednosti svojstava izvorišta ne postanu dostupne.

Zadatak 220
Objasnite razliku između izvedenih i nasljednih svojstava. Kako se izvedena i nasljedna svojstva spremaju na stog tijekom parsiranja od vrha prema dnu?

Vrijednost nasljednog svojstva znaka desne strane produkcije računa se na temelju vrijednosti svojstava ostalih znakova koji su na lijevoj ili desnoj strani produkcije. Vrijednost izvedenog svojstva nezavršnog znaka lijeve strane produkcije računa se na temelju vrijednosti svojstava ostalih znakova koji su na lijevoj ili desnoj strani produkcije.
Parser na stog stavlja oznake semantičkih akcija. Zajedno s oznakom semantičke akcije na stog se stavljaju vrijednosti svojstava te oznake.

Zadatak 221
Navedite zadatke semantičkog analizatora.

1. Određivanje značenja složenim sintaksnim strukturama
2. Provjera semantičkih pravila
3. Postupci pretvorbe vrijednosti obilježja

4. Popunjavanje tablice znakova (vrsta identifikatora, podatkovni tip i djelokrug varijable)
5. Prenošenje vrijednosti svojstava sintaksnih cjelina ostalim dijelovima jezičnog procesora putem stoga/sintaksnog stabla
6. Prenošenje vrijednosti obilježja po sintaksnom stablu
7. Izravnavanje sintaksnog stabla
8. Nadziranje pogrešaka
9. Provjera podudarnosti podatkovnih tipova varijabli
10. Provjera tijeka izvođenja programa
11. Provjera ispravnosti definicije
12. Provjera imena

Zadatak 224

Izgradite potisni automat za zadanu atributnu prijevodnu gramatiku. Za sve akcije *Zamijeni* prikazati stanje na stogu neposredno prije i neposredno poslije primjene akcije.

(1) $S \to a_p\, b_q\, A_r\, \{X_v\}$ $\quad v \leftarrow p \times q + r$

(2) $A_p \to a_q\, B_r$ $\quad p \leftarrow q + r$

(3) $B_p \to c_q$ $\quad p \leftarrow q$

Dodajemo akcijske znakove

$\{R\}_{x1,\,x2,\,x3,\,x4}$ $\quad x4 \leftarrow x1 \cdot x2 + x3$

$\{Zbroj\}_{x1,\,x2,\,x3}$ $\quad x3 \leftarrow x1 + x2$

Novi oblik gramatike:

$S \to a_p\, b_q\, A_r\, \{R\}_{x1,\,x2,\,x3,\,x4}\{X_v\}$ $\quad x1 \leftarrow p,\ x2 \leftarrow q,\ x3 \leftarrow r,\ v \leftarrow x4$

$A_p \to a_q\, B_r\{Zbroj\}_{x1,\,x2,\,x3}$ $\quad x1 \leftarrow q,\ x2 \leftarrow r,\ p \leftarrow x3$

$B_p \to c_q$ $\quad p \leftarrow q$

Tablica potisnog automata:

	a	b	c	$\perp$
S	(1)	*Odbaci*	*Odbaci*	*Odbaci*
A	(2)	*Odbaci*	*Odbaci*	*Odbaci*
B	*Odbaci*	*Odbaci*	(3)	*Odbaci*
b	*Odbaci*	(3)	*Odbaci*	*Odbaci*
∇	*Odbaci*	*Odbaci*	*Odbaci*	*Prihvati*
$\{X\}$	*Izlaz*($\{X\}$); *Izvuci*; *Zadrži*;			
$\{R\}$	*Izlaz*($\{R\}$); *Izvuci*; *Zadrži*;			
$\{Zbroj\}$	*Izlaz*($\{Zbroj\}$); *Izvuci*; *Zadrži*;			

(1) *Zamijeni*($\{X\}\{Zbroj\}A\{Umnožak\}b$); *Pomakni*;

(2) *Zamijeni*($\{Zbroj\}B$); *Pomakni*;

(3) *Izvuci*: *Pomakni*;

5 Potpora izvođenju ciljnog programa

Zadatak 227
Opišite algoritam gradnje lanca kazaljki nelokalnih imena i vektora dubine gniježđenja kod statičkog pravila djelokruga ugniježđenih procedura.

Način određivanja vrijednosti kazaljke nelokalnih imena pozvane procedure i gradnja vektora ovise o dubini gniježđenja pozivajuće i pozvane procedure. Algoritam zasebno obrađuje dva različita slučaja:

1. Pozvana procedura deklarirana je naredbama pozivajuće procedure

   → Kazaljci nelokalnih imena pozvane procedure odredi se vrijednost tako da pokazuje na opisnik pozivajuće procedure

   → Vektoru dubine gniježđenja doda se novi element $i = j + 1$ koji pokazuje na opisnik pozvane procedure (dubina je gniježđenja pozvane procedure $i = j + 1$)

2. Pozvana procedura deklarirana je naredbama procedure koja ugnježđuje pozvanu i pozivajuću proceduru

   → Kazaljci nelokalnih imena pozvane procedure odredi se vrijednost tako da pokazuje na opisnik procedure koji se dohvati slijeđenjem $j - (i - 1)$ kazaljki nelokalnih imena počev od kazaljke pozivajuće procedure

   → Vrijednost elementa vektora dubine gniježđenja na mjestu i spremi se u opisnik pozvane procedure. Element vektora na mjestu i poprimi vrijednost kazaljke na opisnik pozvane procedure.

Zadatak 228
Objasnite povezivanje imena izvornog programa i objekata ciljnog programa te relaciju okoline i relaciju stanja.

Način povezivanja imena izvornog programa, podatkovnih objekata ciljnog programa i vrijednosti podatkovnih objekata određen je vrijednostima dviju relacija: *relacija okoline* pridružuje imenima izvornog programa podatkovne objekte ciljnog programa, a *relacija stanja* pridružuje podatkovnom objektu vrijednost.

Zadatak 229
Ukratko definirajte relaciju okoline i relaciju stanja.

Relacija okoline i relacija stanja relacije su čije vrijednosti određuju način povezivanja imena izvornog programa, podatkovnih objekata ciljnog programa i vrijednosti podatkovnih objekata.

Zadatak 230
Objasnite načine ostvarenja dinamičkog pravila djelokruga.

Ako se primjenjuje dinamičko pravilo djelokruga, onda se važeće deklaracije nelokalnih imena nasljeđuju iz pozivajuće procedure. Dinamičko pravilo djelokruga moguće je ostvariti na dva načina:

1. Pretraživanje po dubini = ako ime nije lokalno definirano, pretražuju se opisnici procedura primjenom lanca kazaljki sve dok se ne pronađe opisnik koji sadrži zadanu deklaraciju imena

2. Pretraživanje statičke memorije = vrijednosti svih lokalno definiranih imena spremaju se u statičku memoriju; prije spremanja vrijednosti lokalnih deklaracija provjeri se sadržaj statičke memorije, ako je zadano ime deklarirano, onda se vrijednosti prethodnih deklaracija sačuvaju u opisniku pozvane procedure; u statičkoj se memoriji prethodne vrijednosti deklaracija zamijene novim vrijednostima; nakon završetka izvođenja pozvane procedure prethodno sačuvane vrijednosti deklaracija prepišu se iz njezinog opisnika u statičku memoriju

Zadatak 231
Opišite mehanizam povratne razmjene vrijednosti parametara procedura te navedite način ostvarenja.

Ako se koristi mehanizam povratne razmjene vrijednosti, onda se u opisnik pozvane procedure zapisuju vrijednosti aktualnih parametara i njihove adrese:

1. Formalni su parametri lokali podaci pozvane procedure. U opisniku pozvane procedure ostavljaju se prazna mjesta za zapis vrijednosti i adresa aktualnih parametara.

2. Pozivajuća procedura odredi vrijednosti i adrese aktualnih parametara i zapiše ih u opisnik pozvane procedure. Pozvana procedura koristi i mijenja isključivo lokalne vrijednosti spremljene u svom opisniku. Nakon što završi njeno izvođenje, pozvana procedura pročita iz svog opisnika vrijednosti aktualnih parametara i njihove adrese. Vrijednosti aktualnih parametara spreme se u memoriju primjenom pročitanih adresa.

Zadatak 232
Navedite i kratko objasnite postupke za određivanje djelokruga deklaracije nelokalnih imena.

Djelokrug deklaracije nelokalnih imena identifikatora određuje se na jedan od sljedećih načina: (1) statički djelokrug bez ugniježđenih procedura - sve su procedure deklarirane isključivo naredbama glavnog programa, a ako identifikator nije deklariran naredbama procedure, potrebno ga je deklarirati naredbama glavnog programa; (2) statički djelokrug ugniježđenih procedura - pravilo se zasniva na pravilu najbliže ugniježđene procedure; (3) dinamički djelokrug - ako naredbe pozvane procedure ne deklariraju ime identifikatora, onda su važeće deklaracije pozivajuće procedure, a postupak određivanja važeće deklaracije nastavlja pretraživanje ostalih pozivajućih procedura.

Zadatak 233
Objasnite način ostvarenja statičkog pravila djelokruga nelokalnih imena ugniježđenih procedura.

Statičko pravilo djelokruga ugniježđenih procedura zasniva se na pravilu najbliže pozivajuće procedure:

1. Djelokrug deklaracije koja je zadana naredbom procedure uključuje sve naredbe te procedure.

2. Ako ime x nije deklarirano naredbama procedure $q()$, onda je važeća deklaracija zadana naredbama ugnježđujuće procedure $p()$ za koju vrijedi: ime x deklarirano je naredbama procedure $p()$ i ne postoji nijedna druga procedura $r()$ za koju vrijedi: ime x deklarirano je naredbama procedure $r()$, procedura $p()$ ugnježđuje proceduru $r()$ i procedura $r()$ ugnježđuje proceduru $q()$.

Zadatak 234
Navedite osnovne načine razmjene ulazno/izlaznih parametara procedura i što se zapisuje u opisnik pozvane procedure prilikom pojedinog načina razmjene.

1. Razmjena vrijednosti = u opisnik pozvane procedure zapisuju se vrijednosti aktualnih parametara
2. Povratna razmjena vrijednosti = u opisnik pozvane procedure zapisuju se vrijednosti aktualnih parametara i njihove adrese
3. Razmjena adrese = u opisnik pozvane procedure zapisuju se adrese aktualnih parametara
4. Razmjena imena = u opisnik pozvane procedure zapisuju se podaci koji se koriste za računanje adrese aktualnih parametara

Zadatak 235
Objasnite pojmove djelokrug deklaracije i životni vijek pridruživanja imena. Što se događa sa životnim vijekom pridruživanja imena prilikom rekurzivnih poziva potprograma?

Područje programa u kojem je važeća deklaracija varijable naziva se djelokrug varijable. Životni vijek procedure započinje izvođenjem njezine početne naredbe, a završava izvođenjem njezine završne naredbe. Priliko rekurzivnih poziva potprograma životni vijek aktiviranog potprograma ugniježđen je u životni vijek pozivajućeg (pot)programa.

Zadatak 236
Objasnite djelokrug deklaracije i navedite moguća pravila definiranja djelokruga deklaracija.

Ako varijabla, procedura, polje ili neki drugi identifikator nije lokalno deklariran naredbama procedure, onda pravila djelokruga određuju njegovu važeću deklaraciju. Djelokrug deklaracije nelokalnih imena identifikatora određuje se na jedan od sljedećih načina: statički djelokrug bez ugniježđenih procedura, statički djelokrug ugniježđenih procedura i dinamički djelokrug.

Zadatak 237
Objasnite vektor dubine gniježđenja i algoritam njegove izgradnje.

Čitanje vrijednosti ulančanih kazaljki nelokalnih imena moguće je izbjeći uporabom vektora dubine gniježđenja. Ako je dubina gniježđenja procedure koja se izvodi n, onda vektor ima n elemenata. Dubini gniježđenja i, $1 \leq i \leq n$, dodjeljuje se element vektora i. Element vektora i kazaljka je koja pokazuje na opisnik posljednje aktivirane procedure dubine gniježđenja i. Ako se želi dohvatiti deklaracija imena a dubine gniježđenja i, onda se čitanjem kazaljke u vektoru na mjestu i izravno dohvati opisnik procedure koja deklarira zadano ime a.

Zadatak 225

Za svaki od četiri načina prenošenja parametara odredite stanja varijabli `j`, `k`, `l` te polja `i` nakon izvođenja zadanog programskog odsječka.

```
int i[3] = {5, 6, 7}, j = 8, k = 2, l = 0;
divmod(int a, int b, int c, int d)
{
    c = a / b;
    d = a % b;
    l = a % b + 1;
}
divmod(i[k], j, k, l);
```

formalni parametri	a	b	c	d
aktualni parametri	i[2]	j	k	l

Zadatak 238

Za prikazani programski odsječak odredite ispis ako se kod poziva potprograma koristi: (a) razmjena vrijednosti, (b) razmjena adresa, (c) razmjena imena i (d) povratna razmjena vrijednosti.

```
varijabla x = 0, y = 3, z = -1;
polje o[3] = 10, o[4] = 20;
    Racunaj(p, q, r) {
        z = p + x
        q = q + 1
        print(p, q, r);
        r = z + q
    }
{
    za x = 3 do 4 {
        Racunaj(o[x], o[3 + x % 2], z);
        print(x, y, z, o[3], o[4]);
    }
}
```

Mehanizam razmjene imena

x	y	z	o[3]	o[4]	p	q	r
0	3	-1	10	20	o[x]	o[3 + x % 2]	z
3	3	-1	10	20	o[3]	o[4]	-1
3	3	13	10	20	o[3]	o[4]	13
3	3	4	10	20	o[3]	o[4]	4
Ispis: 10 3 4							
3	3	24	10	20	o[3]	o[4]	24
Ispis: 3 3 24 10 20							
4	3	24	10	20	o[x]	o[3 + x % 2]	z
4	3	24	10	20	o[4]	o[3]	24
4	3	4	10	20	o[4]	o[3]	4
Ispis: 20 4 4							
4	3	14	10	20	o[4]	o[3]	14
Ispis: 4 3 14 10 20							

Zadatak 240

Za zadani program prikažite sadržaj opisnika procedura u trenutku prije izvođenja naredbe `return x + y + 1` ako se koristi: (a) statičko pravilo djelokruga, (b) dinamičko pravilo djelokruga.

```
Glavni()
    int y = 3;
    def Z(a)
        int r = 5
        def X(x) {
            return x+y+1;
        }
        def Y(y){
            return X(y)+1;
        }
    {
        Y(a);
    }
{
    print Z(y+1)
}
```

Ako se koristi statičko pravilo djelokruga ugniježđenih procedura, vrijedi da upravljačka kazaljka procedure pokazuje na proceduru koja ju je pozvala, a kazaljka nelokalnih imena procedure pokazuje na proceduru u kojoj je trenutna procedura ugniježđena.

Ako se koristi dinamičko pravilo djelokruga ugniježđenih procedura, vrijedi da upravljačka kazaljka procedure pokazuje na proceduru koja ju je pozvala, a kazaljka nelokalnih imena procedure također pokazuje na proceduru koja ju je pozvala.

Statičko pravilo djelokruga ugniježđenih procedura

Opisnik programa `Glavni()`
Lokalni podaci `y = 3`
Opisnik potprograma `Z()`
Ulazni parametri `a = 4`
Upravljačka kazaljka (*pokazuje na* `Glavni`)
Kazaljka nelokalnih imena (*pokazuje na* `Glavni`)
Lokalni podaci `r = 5`
Opisnik potprograma `Y()`
Ulazni parametri `y = 4`
Upravljačka kazaljka (*pokazuje na* `Z`)
Kazaljka nelokalnih imena (*pokazuje na* `Z`)
Lokalni podaci (*nema ih*)
Opisnik potprograma `X()`
Ulazni parametri `x = 4`
Upravljačka kazaljka (*pokazuje na* `Y`)
Kazaljka nelokalnih imena (*pokazuje na* `Z`)
Lokalni podaci (*nema ih*)

Dinamičko pravilo djelokruga ugniježđenih procedura

Opisnik programa `Glavni()`
Lokalni podaci `y = 3`
Opisnik potprograma `Z()`
Ulazni parametri `a = 4`
Upravljačka kazaljka (*pokazuje na* `Glavni`)
Kazaljka nelokalnih imena (*pokazuje na* `Glavni`)
Lokalni podaci `r = 5`
Opisnik potprograma `Y()`
Ulazni parametri `y = 4`
Upravljačka kazaljka (*pokazuje na* `Z`)
Kazaljka nelokalnih imena (*pokazuje na* `Z`)
Lokalni podaci (*nema ih*)
Opisnik potprograma `X()`
Ulazni parametri `x = 4`
Upravljačka kazaljka (*pokazuje na* `Y`)
Kazaljka nelokalnih imena (*pokazuje na* `Y`)
Lokalni podaci *nema ih*

Zadatak 243

Prikažite razliku između razmjene parametara primjenom mehanizma razmjene adresa i mehanizma razmjene imena na sljedećem programu:

```
var x = 0
polje A = {10, 20} // A[0]=10, A[1]=20
P(a) {
    x = 1
    a = 100
    print((A[0], A[1])
}
{
    P(A[x])
}
```

Mehanizam razmjene adresa

- U opisniku pozvane procedure ostavljaju se praznam mjesta za adrese aktualnih parametara
- Pozivajuća procedura izračuna adrese aktualnih parametara, zapiše ih u opisnik pozvane procedure koja ih potom koristi

x	a	A[0]	A[1]
0	A[x]	10	20
1	A[0]	100	20

Ispis: `100 20`

Mehanizam razmjene imena

- U opisniku pozvane procedure ostavljaju se prazna mjesta koja se potom koriste za računanje adresa aktualnih parametara
- Pozivajuća procedura izračuna adrese aktualnih parametara, zapiše ih u opisnik pozvane procedure koja iznova računa adresu aktualnog parametra za svaki dohvat vrijednosti

x	a	A[0]	A[1]
0	A[x]	10	20
1	A[1]	10	100

Ispis: `10 100`

6 Generiranje međukôda

Zadatak 245
Navedite i kratko opišite linearne oblike međukôda.

Međukôd linearnog oblika čine troadresne naredbe. Troadresne su naredbe zapis izravnatog sažetog sintaksnog stabla ili izravnog grafa bez petlji. Imamo:

- *Troadresne naredbe ostvarene četvorkama* - koriste četiri polja: polje operatora, dva polja operanda i polje rezultata
- *Troadresne naredbe ostvarene trojkama* - koriste se tri polja: polje operatora i dva polja operanda
- *Troadresne naredbe ostvarene neizravnim združenim trojkama* - združivanjem istovjetnih trojki i korištenjem vektora izvođenja učinkovitije se koristi memorijski prostor

Zadatak 246
Navedite osnovne razine međukôda i objasnite namjenu svake razine.

Međukôdom više razine započinje sinteza ciljnog programa. U međukôdu više razine ostaju sačuvane strukture petlji i indeksa polja izvornog programa. Međukôd više razine pogodan je za analizu tijeka izvođenja programa, analizu toka podataka, analizu zavisnosti podataka i analizu pseudonima. Međukôd više razine najčešće je grafičkog oblika (sažeto sintaksno stablo, izravni graf bez petlji, graf zavisnosti programa).
Međukôd srednje razine čine pojednostavljene naredbe izvornog jezika koje sliče strojnim naredbama, simbolička imena indentifikatora izvornog programa i simbolička imena privremenih varijabli. Postupci optimiranja izvođenja procedura, tijeka izvođenja programa, redoslijeda izvođenja naredbi i računanja izraza koriste međukôd srednje razine.
Međukôd niže razine čine naredbe slične naredbama strojnog jezika. Većina naredbi međukôda niže razine prevodi se u jednu strojnu naredbu. Ako je naredbu međukôda niže razine moguće prevesti na više različitih načina, tijekom generiranja i optimiranja ciljnog programa izabire se najpovoljniji način. Naredbe međukôda niže razine ne koriste simbolička imena identifikatora (vrijednosti se dohvaćaju primjenom registara ili adresa memorijskih ćelija). Međukôd niže razine pogodan je za razne postupke strojno zavisnog optimiranja.

Zadatak 247
Objasnite graf zavisnosti.

Graf zavisnosti oblik je međukôda prilagođen postupcima optimiranja. Čine ga dva grafa, graf zavisnosti upravljačkog tijeka i graf zavisnosti podataka. Graf zavisnosti upravljačkog tijeka sadrži podatke o zavisnosti izvođenja naredbi. Četiri su vrste zavisnosti podataka: unaprijedna zavisnost f, unazadna zavisnost a, zavisnost odredišta o i zavisnost izvorišta i.

Zadatak 249

Navedite tri oblika međukôda te objasnite općeniti izgled međukôda svakog oblika.

Grafički oblici međukôda su sažeto sintaksno stablo i izravni graf bez petlji. Listovi su sažetog sintaksnog stabla operandi, a unutrašnji su čvorovi operatori i ključne riječi. Kod izravnog grafa prije definiranja novog čvora provjerava se skup prethodno definiranih čvorova. Ako je čvor prethodno definiran, u graf se dodaje samo grana na postojeći čvor.
Postfiksni sustav oznaka (postfiksni oblici međukôda) nastaje obilaskom čvorova sažetog sintaksnog stabla po dubini. Potpuno izravnavanje postiže se spremanjem operanada i operacija na postisni stog.
Međukôd linearnog oblika čine troadresne naredbe.

Zadatak 257

Navedite tri oblika međukôda.

Po svom se obliku međukôd dijeli na grafički, postfiksni i linearni.

Zadatak 261

Izgradite atributnu prijevodnu gramatiku koja generira troadresne naredbe za računanje logičkih izraza koji sadrže operator $\wedge$, $\vee$ i $\neg$.

$S_{kod} \rightarrow E_{ime1,\,kod1}$ {kod = kod1}

$E_{ime1,\,kod1} \rightarrow \neg\, E_{ime2,\,kod2}$ {ime1 = NovoIme(); kod1 = Generiraj(kod2 || ime1 ":= not" ime2}

$E_{ime1,\,kod1} \rightarrow E_{ime2,\,kod2} \wedge E_{ime3,\,kod3}$
{ime1 = NovoIme(); kod1 = Generiraj(kod2 || kod3 || ime1 ":=" ime2 "and" ime3)}

$E_{ime1,\,kod1} \rightarrow E_{ime2,\,kod2} \vee E_{ime3,\,kod3}$
{ime1 = NovoIme(); kod1 = Generiraj(kod2 || kod3 || ime1 ":=" ime2 "or" ime3)}

$E_{ime1,\,kod1} \rightarrow T_{ime2}$ {ime1 = 1; kod = Generiraj("");}

$E_{ime1,\,kod1} \rightarrow F_{ime2}$ {ime1 = 0; kod = Generiraj("");}

7 Generiranje ciljnog programa

Zadatak 263
Objasnite generiranje ciljnog programa na temelju postfiksnog sustava oznaka.

Ako je međukôd zapisan primjenom postfiksnog sustava oznaka, onda se osim generiranja naredbi ciljnosg programa izravnava sintaksno stablo primjenom potisnog stoga. Na temelju naredbe izvornog programa generira se niz znakova postfiksnog sustava. Generator ciljnog programa čita znakove slijedno s lijeva na desno i primjenjuje jednu od akcija:

1. Stavi pročitani znak međukôda na vrh stoga i pomakni glavu za čitanje na sljedeći znak.
2. Uzmi s vrha stoga zadnji broj operanda, generiraj naredbe ciljnog programa i stavi rezultirajući operand na vrh stoga.

Zadatak 264
Opišite postupak izrade adresa naredbama.

Generator ciljnog programa izrađuje memorijske adrese naredbama ciljnog programa. Adrese naredbi računaju se primjenom brojača. Veličina brojača povećava se za veličinu generirane naredbe izražene u oktetima. postupak izrade adresa naredbi koje upravljaju tijekom izvođenja programa koristi dvije liste:

- Lista unazadnih adresa: čine ju zapisi simboličkih programskih oznaka i memorijskih adresa naredbi kojima su te oznake dodijeljene
- Lista unaprijednih adresa: čine ju zapisi programskih oznaka i kazaljke koje pokazuju na generirane naredbe što koriste koriste te oznake

Zadatak 265
Općenito definirajte ulaze i izlaze iz programa semantičkog analizatora i generatora ciljnog programa ako su oba programa ostvareni kao zasebni prolazi jezičnog procesora.

- Generator ciljnog programa: ulaz sintaksno stablo, izlaz generirani program
- Semantički analizator: ulaz sintaksno stablo, izlaz podatci potrebni za generiranje atributnog sintaksnog stabla / niza naredbi stogovnog stroja / niza slijednih naredbi

Zadatak 266
Opišite Chaitinov heuristički postupak za bojanje grafa zavisnosti simboličkih i stvarnih registara.

U grafu zavisnosti odredi se čvor koji ima manji broj susjeda od broja boja R (R ukupan broj boja za bojanje grafa). Izabrani čvor i sve njegove grane izuzmu se iz grafa zavisnosti. Na stog se spremi izuzeti čvor zajedno s listom svojih susjeda. Postupak izuzimanja čvorova nastavlja se izmijenjenim grafom koji ima jedan čvor manje. Ako se izuzmu svi čvorovi, onoda je moguće obojati graf s R boja, inače u grafu

zavisnosti ostaju čvorovi s R ili više susjeda. Na temelju izabrane funkcije odlučivanja izabere se jedan od simboličkih registara, a njegova vrijednost spremi se u memoriju. Promijeni se ciljni program tako da se varijabli koja je spremljena u izabranom registru pristupa izravno u memoriji. Iz grafa zavisnosti izuzme se čvor pridružen tom simboličkom registru i sve njegove grane. Postupak bojanja grafa nastavlja se s izmijenjenim grafom koji ima jedan čvor manje.

Zadatak 267
Opišite algoritam generiranja ciljnog programa na osnovi troadresnih naredbi.

1. Odredi mjesto M zapisa rezultata R. Prednost se daje registrima, no nije li moguće spremiti u registar, sprema se u memoriju.

2. U opisniku podataka pronađi sva mjesta p gdje je spremljena vrijednost varijable P. Prednost se daje registrima. Ako je mjesto p različito od mjesta zapisa rezultata M izabranog u koraku (1), onda se genetrira naredba `MOVE p, M`.

3. U opisniku podataka pronađi sva mjesta d gdje je spremljena vrijednost varijable D. Prednost se daje registrima. Generira se naredba: OP d, M. U opisnik podataka zapiše se da je varijabla R spremljena na mjestu M. Ako je M registar, onda se u opisnik registara zapiše da registar M sadrži vrijednost varijable R. U opisnike se zapiše da ostala mjesta ne sadrže vrijednost varijable R jer je naredba `R := P op D` promijenila njenu vrijednost.

4. Ako se nakon izvedene naredbe`R := P op D` vrijednosti varijabli P i D ne koriste u nastavku programa i ako su njihove vrijednosti u registrima, onda se u zapisnike zapiše da registri više ne sadrže vrijednosti tih varijabli.

Zadatak 268
Navedite elemente strukture generatora ciljnog programa.

Generator ciljnog programa definira se ulazom i izlazom, a čine ga postupci izrade adrese, izbora naredbe, izbora redoslijeda izvođenja naredbi i dodjele registara podacima.

Zadatak 269
Objasnite kako se dobiva i boji graf zavisnosti simboličkih i stvarnih registara te kako se dodjeljuju stvarni registri.

1. Odrede se mrežice (unije *du*-lanaca iste varijable koje imaju zajedničkih varijabli)

2. Mrežicama se dodijele simbolički registri (opće su namjene, neograničene veličine, omogućuju spremanje bilo kojeg tipa podatka, ima ih beskonačno mnogo)

3. Gradi se graf zavisnosti simboličkih i stvarnih registara

4. Ako generator raspolaže s R stvarnih registara procesora, onda se čvorovi grafa zavisnosti boje s R različitih boja, tako da dva susjedna čvora nisu obojana istom bojom

5. Stvarni registar dodijeli se onim simboličkim registrima koji su obojani istom bojom

8 Priprema izvođenja ciljnog programa

Zadatak 279
Navedite razradbu jezičnih procesora s obzirom na stupanj pripremljenosti ciljnog programa za izvođenje.

Ovisno o stupnju pripremljenosti ciljnog programa za izvođenje, jezični se procesori dijele na:

- *Spremi-i-pokreni jezični procesori*
- *Generatori izvodivog ciljnog programa*
- *Generatori premjestivog ciljnog programa*
- *Generatori zasebnih dijelova programa*

9 Optimiranje

Zadatak 280
Opišite što se nastoji utvrditi analizom toka podataka.

Analiza toka podataka izravni je nastavak analize tijeka izvođenja programa. Na temelju grafa tijeka izvođenja programa analizira se način uporabe podataka u programu primjenom iterativnog ili eliminacijskog postupka.

Zadatak 281
Objasnite razliku između strojno nezavisnog i strojno zavisnog optimiranja.

Strojno nezavisni postupci optimiraju programske strukture bliske strukturama višeg programskog jezika (programske petlje, složene naredbe grananja, procedure, pristup složenim apstraktnim tipovima podataka...). *Strojno zavisni postupci* optimiraju programske strukture bliske strukturama strojnog jezika.

Zadatak 282
Nabrojite komponente koje čine analizu izvođenja programa.

Analizu izvođenja programa čine analiza tijeka izvođenja programa, analiza toka podataka, analiza zavisnosti podataka i analiza pseudonima.

Zadatak 283
Opišite postupak optimiranja petlji kod međukôda niže razine i ciljnog programa.

Optimiranje petlji obuhvaća više različitih preinaka:

- Izuzimanje praznih petlji
- Invertiranje = postupak zamjene naredbe uvjetnog i bezuvjetnog grananja samo jednom naredbom grananja na kraju petlje
- Izravnavanje = izuzimanje naredbi grananja i ponavljanje naredbi njezinog tijela onoliko puta koliko se puta izvodi petlja

Zadatak 284
Opišite analizu tijeka izvođenja programa.

Analiza tijeka izvođenja programa ispituje upravljačku strukturu međukôda definiranu naredbama petlji, grananja itd. Dva su osnovna pristupa analizi tijeka izvođenja programa: analiza dominacije i analiza strukture. Obje analize grade graf tijeka izvođenja. Analiza dominacije određuje strukturu programskih petlji. Analiza strukture gradi upravljačko stablo na temelju raščlambe grafa tijeka izvođenja programa.

Zadatak 285

Nabrojite i kratko opišite postupke optimiranja međukôda srednje razine.

Budući da međukôd srednje razine čine pojednostavljene naredbe izvornog jezika koje sliče strojnim naredbama, moguće ga je optimirati primjenom različitih postupaka:

- Optimiranje procedura = pretvorba rekurzivnih poziva procedura u petlje i uvišestručavanje procedura
- Optimiranje petlji = ovim se postupkom optimira provjera granica indeksa polja i uporaba spregnutih varijabli (vrijednost varijable tijekom izvođenja petlje postupno se smanjuje ili povećava za neku konstantnu vrijednost)
- Optimiranje izraza = unaprijedno računanje izraza konstantne vrijednosti, izlučivanje izraza izvan petlje, izuzimanje istovjetnih izraza i mrtvih izraza, razlaganje i pojednostavljenje izraza, ...

10 Ispitni zadaci - MI

Zadatak 2022

Ako imamo $X_{n_5,i_1,i_2} \to Y_{n_6}\, Z_{i_3}\, V_{i_4,n_7,n_8}\, W_{n_9}$, n_7 može se izračunati na temelju čega? i_1, i_2, i_3 i i_4 su izvedena svojstva, n_5, n_6, n_7, n_8 i n_9 su nasljedna svojstva.

iz primjera 4.3 imamo:

$$X_{n_1,i_1,i_2} \to Y_{n_4}\, Z_{i_5}\, V_{i_6,n_7,n_8}\, W_{n_9}$$

$n_4 = f(n_1)$ $\quad$ $n_7, n_8 = f(n_1, n_4, i_5)$ $\quad$ $n_9 = f(n_1, n_4, n_7, n_8, i_5, i_6)$ $\quad$ $i_1, i_2 = f(n_1, n_4, n_7, n_8, n_9, i_5, i_6)$
pa je rješenje: $n_7 = f(n_5, n_6, i_3)$

Zadatak 2015

Za slijedeću kontektsno neovisnu gramatiku
$S \to aABc \,|\, cBAb$
$A \to aA \,|\, \varepsilon$
$B \to bB \,|\, c$
izračunajte vrijednost relacije *ReduciranZnakom* za završni znak c.

SLIJEDI$(S) = \{\perp\}$, SLIJEDI$(B) = \{a, b, c\}$
imamo relacije *ReduciranZnakom*(c, a), *ReduciranZnakom*(c, b), *ReduciranZnakom*(c, c), *ReduciranZnakom*$(c, \perp)$

Zadatak 2015

Za slijedeću kontekstno neovisnu gramatiku
$S \to pAmC$ $\quad$ $S \to bA$ $\quad$ $A \to dSa$ $\quad$ $C \to dA$ $\quad$ $A \to e$
izračunajte vrijednost relacije *IspodZnaka* za završni znak p.

imamo relacije *IspodZnaka*(p, d) i *IspodZnaka*(p, e)

Zadatak 2017

Na raspolaganju nam je n viših programskih jezika $L_1, \ldots, L_n$, računalo čiji je strojni jezik označen sa b te jezični procesor $\text{JP}_b^{L_1 \to b}$ koji na tom računalu prevodi L_1 u b. Na istom računalu želimo imati sve prevoditelje viših jezika iz jednog u drugi, tj. sve $\text{JP}_b^{L_i \to L_j}$ za $i, j \in \{1, \ldots, n\}, i \neq j$. Odredite minimalni broj procesora oblika $\text{JP}_{L_k}^{L_i \to L_j}$ koje je potrebno napisati u višim jezicima tako da se njihovom primjenom (nadovezivanjem, prevođenjem) mogu izgraditi svi traženi procesori. (Želimo imati mogućnost prevođenja $L_i \to L_j$, a ne sve moguće procesore. Dakle, minimalan broj procesora kako bi njihovom kombinacijom mogli ostvariti sva spomenuta prevođenja.)

Minimalan je broj procesora n. To su procesori oblika

$$L_1 \to L_2 \;-\; L_2 \to L_3 \;-\; \cdots \;-\; L_{n-1} \to L_n \;-\; L_n \to L_1$$

Jezik izgradnje jezik je L_1 (prevođenjem procesorom $\text{JP}_b^{L_1 \to b}$ prevodimo jezik izgradnje u jezik b).

Zadatak 2013

Za jezik D implementirali smo samoprevoditelj koji prevodi u izvršivi jezik x. Želimo dobiti izvodivu inačicu prevoditelja iz jezika D u jezik x, a na raspolaganju uz navedeni samoprevoditelj imamo sljedeće jezične procesore: $\mathrm{JP}_D^{A\to C}$, $\mathrm{JP}_y^{C\to A}$, $\mathrm{JP}_y^{C\to y}$, $\mathrm{JP}_x^{C\to B}$, $\mathrm{JP}_A^{B\to D}$, $\mathrm{JP}_C^{D\to C}$, $\mathrm{JP}_x^{B\to z}$. Izvodivi su jezici x, y i z. Prikažite neki najkraći postupak kojim možemo ostvariti zadani cilj.

Dodatno imamo prevoditelj $\mathrm{JP}_D^{D\to x}$. Najkraći je postupak :

$$\mathrm{JP}_D^{D\to x} \xrightarrow{\mathrm{JP}_D^{D\to C}} \mathrm{JP}_C^{D\to x} \xrightarrow{\mathrm{JP}_y^{C\to y}} \mathrm{JP}_y^{D\to x}$$

Zadatak Teorijska pitanja

Ovo je neka teorija koja je bitna u zadacima.

- LR stavka produkcija je gramatike koja ima oznaku točke na proizvoljnom mjestu svoje desne strane uključujući krajnje lijevo i krajnje desno mjesto. U slučaju ε-produkcije, $A \to \varepsilon$, LR-stavka je $A \to \bullet$.
- PRIMIJENI$(A \to \alpha)$ = ZAPOČINJE$(\alpha) \cup$ SLIJEDI(A), ako je $A \to \alpha$ prazna produkcija, inače PRIMIJENI$(A \to \alpha)$ = ZAPOČINJE(α)
- Relacija *ReduciranZnakom*(A, x) vrijedi ako je znak A početni nezavršni znak gramatike, a znak x oznaka kraja niza, $\perp$
- Kontekstno neovisna gramatika jest operatorska ako i samo ako produkcije nisu ε-produkcije i ni na jednom mjestu desne strane produkcije dva susjedna znaka nisu nezavršna.
- Ako je M broj različitih operatora u programu, a N broj naredbi, veličina Co–No tablice je M^2 (broj redaka jednak je broju stupaca jednak je broju operatora)
- Ulazna/izlazna gramatika dio je prijevodne gramatike koja uključuje samo ulazne/izlazne završne znakove $G_{\text{ulazna/izlazna}} = (V, T_{\text{ulazni/izlazni}}, P, S)$
- Početna vrijednost nasljednog svojstva početnog nezavršnog znaka definirana je unaprijed zajedno s produkcijama gramatike.
- Ako je parametar potprograma varijabla koja je pridružena nasljednom svojstvu, onda se razmjenjuje *vrijednost* varijable. Ako je parametar potprograma varijabla koja je pridružena izvedenom svojstvu, onda se razmjenjuje *adresa* varijable.
- Redoslijed zapisa znakova u tablici uniformnih znakova odgovara redoslijedu leksičkih jedinki u izvornom programu.
- Interpretatori: redoslijed prevođenja određen je redoslijedom izvođenja

11 Ispitni zadaci - ZI

Zadatak Teorijska pitanja
Ovo je neka teorija koja je bitna u zadacima.

- Analizu izvođenja programa čini poredani slijed niza:
  1. analiza tijeka izvođenja programa
  2. analiza toka podataka
  3. analiza zavisnosti podataka
  4. analiza pseudonima
  5. postupci pretvorbe
- Početna konfiguracija potisnog automata parsera od vrha prema dnu za zadanu L-atributnu prijevodnu gramatiku:
  - početni nezavršni znak gramatike
  - početne vrijednosti nasljednih svojstava početnog nezavršnog znaka
  - kazaljke koje pokazuju na mjesto zapisa vrijednosti izvedenih svojstava početnog nezavršnog znaka
  - ostatak stoga
- Statička provjera vrijednosti obilježja izvodi se tijekom semantičke analize.
- Ako je zadana troadresna naredba `Oznaka5: a := a + 1`, onda se `Oznaka5` zapiše u listu *unazadnih adresa*.
- Za premjestivi ciljni program nisu u potpunosti izrađene adrese podataka i naredbi.
- Učinkovita uporaba naredbenog cjevovoda zahtjeva grupiranje nezavisnih naredbi.
- Oblici međukôda:
  - sažeto sintaksno stablo
  - izravni graf bez petlji
  - postfiksni sustav oznaka
  - troadresne naredbe
  - statičko jednostruko pridruživanje
  - graf zavisnosti
- U grafu tijeka izvođenja programa, ako se čvor a nalazi na svakom putu od početnog čvora do čvora b, onda je čvor a dominator čvora b.

- Ako je u stablu dominacije čvor a roditelj (neposredni prethodnik) čvora b, onda je u grafu tijeka izvođenja program čvor a neposredni dominator čvora b.
- U metodi rekurzivnog spusta za L-atributnu prijevodnu gramatiku, znakovima gramatike pridružuju se potprogrami, a svojstvima znakova pridružuju se programske varijable.
- Prostorno bliskim pristupima podacima osiguran je unaprijedni dohvat podataka u priručnu memoriju.
- Pri traženju slobodnog segmenta memorijskog prostora, pretpostavimo da nijedan slobodni segment pojedinačno nije dovoljno velik za traženu memoriju. U tom se slučaju najprije pokušava provesti postupak združivanja susjednih slobodnih segmenata.
- Dva su osnovna načina traženja slobodnog segmenta memorijskog prostora: potraga za prvim slobodnim segmentom dovoljne veličine i potraga za slobodnim segmentom koji je po svojoj veličini veći, ali najbliži veličini tražene memorije.
- Generiranje ciljnog programa na temelju postfiksnog sustava oznaka: Ako se u međukôdu pročita operator, onda generator primijeni akciju: uzmi s vrha stoga zadani broj operanada, generiraj naredbe ciljnog programa i stavi rezultirajući operand na vrh stoga.
- U metodi rekurzivnog spusta za L-atributnu prijevodnu gramatiku, ako je varijabli pridruženo nasljedno svojstvo, razmjenjuje se njezina vrijednost, a ako je varijabli pridruženo izvedeno svojstvo, razmjenjuje se njezina adresa.
- Optimiranje zamjenom naredbe poziva procedure naredbama njezinog tijela u pravilu smanjuje vrijeme izvođenja i povećava veličinu ciljnog programa.
- Pri izgradnji potisnog automata za atributnu prijevodnu gramatiku, temeljem produkcije $A \to \xi b \phi \alpha$, pri čemu su ξ i ϕ izlazni znakovi gradi se akcija $Izlaz(\xi, \phi)$; $Zamijeni(\alpha^r)$; $Pomakni$;
- Izvođenje naredbi za koje vrijedi da se vrijednost relacije okoline ne mijenja određuje životni vijek pridruživanja imena.
- Analizu pseudonima čine dva dijela - *skupljač pseudonima* i *prenositelj pseudonima.*
- Strojno nezavisni program virtualnog stroja u strojni program prevodi jezični *postprocesor.*
- Opisnik procedure čine:
  1. Vrijednosti ulaznih parametara procedure
  2. Upravljačka kazaljka (uvijek pokazuje na pozivajuću proceduru)
  3. Kazaljka nelokalnih imena
  4. Lokalni podaci

- Za podjelu jezičnih procesora s obzirom na stupanj pripremljenosti ciljnog programa za izvođenje vidi zadatak 279.
- Generator ciljnog programa čine:
  1. Postupci izrade adrese (podacima i naredbama)
  2. Izbor naredbe
  3. Izbor redoslijeda izvođenja naredbi
  4. Dodjela registara podacima
- Stablo dominacije gradi se na temelju grafa tijeka izvođenja.
- Dinamička provjera vrijednosti obilježja izvodi se tijekom izvođenja ciljnog programa.
- Generator ciljnog programa koristi listu unaprijednih adresa i listu unazadnih adresa.
- Za premjestivi ciljni program vrijedi:
  - čine ga strojne naredbe zapisane nizom nula i jedinica
  - adrese podataka i naredbi nisu u potpunosti izrađene
  - neizrađene su adrese relativni pomaci od nulte adrese
  - vrijednosti adresa određuju se tijekom punjenja radne memorije
  - postupak dorade adresa izvodi *program punitelj*
  - *program povezivač* povezuje zasebno prevedene procedure
- Prilikom generiranja ciljnog programa na temelju postfiksnog sustava oznaka, izravnavanje sintaksnog stabla ostvaruje se primjenom potisnog stoga.
- Dio opisnika procedure po kojem se razlikuju statičko i dinamičko pravilo djelokruga jest kazaljka nelokalnih imena.
- Tijekom analize struktura pri analizi tijeka izvođenja programa, analiza strukture obilazi graf tijeka izvođenja programa, traži podgrafove uzorke, zamijeni ih jednim zamjenskim čvorom i gradi *upravljačko stablo*.
- Gradimo atributnu prijevodnu gramatiku koja generira troadresne naredbe za računanje logičkih izraza. Produkciju $E_{ime1,\,kod1} \rightarrow \neg E_{ime2,\,kod2}$ ima smisla proširiti sljedećim akcijskim znakovima: {ime1 = NovoIme(); kod1 = Generiraj(kod2 || ime1 ":= not" ime2}
- U analizi dominacije koju provodimo nad grafom tijeka izvođenja programa, $\mathrm{dom}(x)$ označava skup dominatora čvora x. Čvor d neposredni je dominator čvora a ako i samo ako za svaki drugi čvor c ($c \neq a$, $c \neq d$) vrijedi: $c \notin \mathrm{dom}(a)$ ili $d \notin \mathrm{dom}(c)$.
  Ovaj se izraz dobije negacijom sljedećeg izraza: Čvor d neposredni je dominator čvora a ako i samo ako ni za jedan drugi čvor c ($c \neq a$, $c \neq d$) ne vrijedi : $c \in \mathrm{dom}(a)$ i $d \in \mathrm{dom}(c)$.

- Četiri vrste zavisnosti podataka:
  1. Unaprijedna zavisnost f - ranija naredba mijenja/definira varijablu koju koristi kasnija naredba
  2. Unazadna zavisnost a - ranija naredba koristi varijablu koja se u kasnijem trenutku mijenja tj. ponovno definira
  3. Zavisnost odredišta o - obje naredbe mijenjaju istu varijablu (ista varijabla je s lijeve strane jednakosti)
  4. Zavisnost izvorišta i - obje naredbe koriste istu varijablu (imaju istu varijablu na desnoj strani jednakosti)
- Parser *Pomakni-Reduciraj* koristi tablice *Pomakni/Reduciraj* i *Stavi*.
- Dio izvornog programa u kojem je važeća deklaracija naziva se djelokrug deklaracije.
- Tri osnovne razine međukôda su međukôd više, niže i srednje razine.
- *Pomakni-Pronađi* parser gradi se izravno na temelju relacija *IspodZnaka* i *ReduciranZnakom*.
- *LR* parser ima tablice $Akcija$ čiji su stupci označeni nezavršnim znakovima (i oznakom kraja niza) i tablicu $NovoStanje$ čiji su stupci označeni nezavršnim znakovima gramatike.

Zadatak ZI 2017

Za programski odsječak odredite ispis ako se kod poziva potprograma koristi...

```
01      x = 0
02      A[2] = {10, 20}
03      P(a) {
04          x = 1
05          a = 100
06          print(A[0], A[1])
07          x = 0
08          a = 101
09      }
10      P(A[x])
11      print(x, A[0], A[1])
```

a) razmjena vrijednosti

n	x	a	A[0]	A[1]	ispis
10	0	10	10	20	
04	1	10	10	20	
05	1	100	10	20	
06	1	100	10	20	10, 20
07	0	100	10	20	
08	0	101	10	20	
11	0		10	20	0, 10, 20

b) razmjena adresa

n	x	a	A[0]	A[1]	ispis
10	0	A[0]	10	20	
04	1	A[0]	10	20	
05	1	A[0]	100	20	
06	1	A[0]	100	20	100, 20
07	0	A[0]	100	20	
08	0	A[0]	101	20	
11	0		101	20	0, 101, 20

c) razmjena imena

n	x	a	A[0]	A[1]	ispis
10	0	A[x]	10	20	
04	1	A[x]	10	20	
05	1	A[x]	10	100	
06	1	A[x]	10	100	10, 100
07	0	A[x]	10	100	
08	0	A[x]	101	100	
11	0		101	100	0, 101, 100

d) povratna razmjena vrijednosti

n	x	a	A[0]	A[1]	ispis
10	0	10	10	20	
04	1	10	10	20	
05	1	100	10	20	
06	1	100	10	20	10, 20
07	0	100	10	20	
08	0	101	101	20	
11	0		101	20	0, 101, 20

U koraku 08 mehanizam povratne razmjene vrijednosti prepiše `a` u `A[x]`, a `x` = 0 (vrijednost se zapravo prepiše u `A[x]`, a gledamo vrijednost koju je `x` imao pri pozivu funkcije s tim parametrom).

Zadatak ZI 2018

Za programski odsječak odredite ispis ako se kod poziva potprograma koristi...

```
01      i = 1
02      b[3] = {5, 6, 7}
03      f(x) {
04          i = 0
05          x = 8
06          print(b[0], b[1], b[2])
07          i = 2
08          x = 9
09      }
10      f(b[i])
11      print(i, b[0], b[1], b[2])
```

a) razmjena vrijednosti

n	i	b[0]	b[1]	b[2]	x	ispis
10	1	5	6	7	6	
04	0	5	6	7	6	
05	0	5	6	7	8	
06	0	5	6	7	8	5, 6, 7
07	2	5	6	7	8	
08	2	5	6	7	9	
11	2	5	6	7		2, 5, 6, 7

b) razmjena adresa

n	i	b[0]	b[1]	b[2]	x	ispis
10	1	5	6	7	b[1]	
04	0	5	6	7	b[1]	
05	0	5	8	7	b[1]	
06	0	5	8	7	b[1]	5, 8, 7
07	2	5	8	7	b[1]	
08	2	5	9	7	b[1]	
11	2	5	9	7		2, 5, 9, 7

c) razmjena imena

n	i	b[0]	b[1]	b[2]	x	ispis
10	1	5	6	7	b[i]	
04	0	5	6	7	b[i]	
05	0	8	6	7	b[i]	
06	0	8	8	7	b[i]	8, 6, 7
07	2	8	6	7	b[i]	
08	2	8	6	9	b[i]	
11	2	8	6	9		2, 8, 6, 9

d) povratna razmjena vrijednosti

n	i	b[0]	b[1]	b[2]	x	ispis
10	1	5	6	7	6	
04	0	5	6	7	6	
05	0	5	6	7	8	
06	0	5	6	7	8	5, 6, 7
07	2	5	6	7	8	
08	2	5	6	7	9	
11	2	5	9	7		2, 5, 9, 7

U koraku 08 mehanizam povratne razmjene vrijednosti prepiše `x` u `b[i]`, `i` = 1.

Zadatak 2017

Za zadanu atributnu prijevodnu gramatiku u pseudokodu sličnom jeziku C napišite parser metodom rekurzivnog spusta. Semantičke akcije $\{Zbroji\}_{r,w,z}$ i $\{Oduzmi\}_{r,w,z}$ primaju argumente r i w te rezultat operacije zapisuju u z.

$S_o \to a\, A_p\, b\, c\, B_{q,r}\, \{Ispisi\}_w$	$o \leftarrow 2;\ p \leftarrow o;\ w \leftarrow q;\ r \leftarrow o$
$S_o \leftarrow b\, A_p\, \{Zbroji\}_{r,w,z}$	$o \leftarrow 2;\ p \leftarrow o;\ w \leftarrow o;\ r \leftarrow o;$
$A_o \to c\, B_{p,q}\, \{Oduzmi\}_{r,w,z}$	$r \leftarrow o;\ q \leftarrow o;\ w \leftarrow p;$
$B_{o,p} \to a\, c$	$o \leftarrow p + 2$

Promjena imena svojstva koja su povezana naredbom preslikavanja:

$S_o \to a\, A_o\, b\, c\, B_{q,o}\, \{Ispisi\}_q$	$o \leftarrow 2;$
$S_o \leftarrow b\, A_o\, \{Zbroji\}_{o,o,z}$	$o \leftarrow 2;$
$A_o \to c\, B_{p,o}\, \{Oduzmi\}_{o,p,z}$	
$B_{o,p} \to a\, c$	$o \leftarrow p + 2$

Dodatna promjena imena svojstava nezavršnog znaka B:

$S_o \to a\, A_o\, b\, c\, B_{q,o}\, \{Ispisi\}_q$	$o \leftarrow 2;$
$S_o \leftarrow b\, A_o\, \{Zbroji\}_{o,o,z}$	$o \leftarrow 2;$
$A_o \to c\, B_{q,o}\, \{Oduzmi\}_{o,q,z}$	
$B_{q,o} \to a\, c$	$q \leftarrow o + 2$

Izrada parsera za preuređenu gramatiku:

```
Glavni(){
    Ulaz.znak = Krajnje lijevi znak niza w;
    Ulaz.vrijednost = Vrijednost krajnje lijevog znaka niza w;

    o = 2
    S(o);

    if (Ulaz.znak != znak_kraja_niza)
        Odbaci();
    else
        Prihvati();
}
```

```
S(o){
    Nasljedno svojstvo o;
    case (Ulaz.znak):
        a:
            {
                Lokalna varijabla q;
                Ulaz.znak = Krajnje lijevi znak niza w;
                Ulaz.vrijednost = Vrijednost krajnje lijevog znaka niza w;
                A(o);
                Ulaz.znak = Krajnje lijevi znak niza w;
                if (Ulaz.znak != b)
                    Odbaci();
                else
                    if (Ulaz.znak != c)
                        Odbaci()
                    else
                        Ulaz.znak = Krajnje lijevi znak niza w;
                        Ulaz.vrijednost = Vrijednost krajnje lijevog znaka niza w;
                        B(&q, o)
                        Ispisi(q);
            }
        b:
            {
                Lokalna varijabla w;
                Ulaz.znak = Krajnje lijevi znak niza w;
                Ulaz.vrijednost = Vrijednost krajnje lijevog znaka niza w;
                A(o);
                Zbroji (o, o, &w);
            }
        Svi ostali znakovi:
            Odbaci();
}
```

```
A(o){
    Nasljedno svojstvo o;
    Lokalna varijabla q, w;
    Ulaz.znak = Krajnje lijevi znak niza w;
    if (Ulaz.znak != c)
        Odbaci();
    else {
        B(&q, o);
        Oduzmi(o, q, &z);
    }
}

B(&q, o){
    Izvedeno svojstvo q;
    Nasljedno svojstvo o;
    q = o + 2;
    Ulaz.znak = Krajnje lijevi znak niza w;
    if (Ulaz.znak != a)
        Odbaci();
    else {
        Ulaz.znak = Krajnje lijevi znak niza w;
        if (Ulaz.znak != c)
            Odbaci();
    }
}
```

Zadatak 2021

Parsiramo L-atributnu prijevodnu gramatiku potisnim automatom.
Neka je zadana sljedeća produkcija:

$$S_{n1,\,i1} \to \{f\}_{n2,\,i2}\, a_x\, A_{i3}\, b_y\, \{h\}_{n3,\,n4,\,i4}\, B_{n5,\,n6,\,n7}$$

$$n2 \leftarrow n1,\ n3 \leftarrow i2,\ n4 \leftarrow x,\ (i1,\, n5) \leftarrow i4,\ n6 \leftarrow i3,\ n7 \leftarrow y$$

gdje je S početni nezavršni znak gramatike. Neka su elementi stoga, počevši od vrha prema dnu, indeksirani s brojevima 0, 1, 2, ... Pod pretpostavkom da se na stogu nalazi početna konfiguracija potisnog automata, što će biti zapisano unutar stoga na indeksu broj 9 nakon jednog koraka rada potisnog automata u kojem pročitamo ulazni završni znak a i primijenimo zadanu produkciju?

Unutar stoga na indeksu 9 zapisano je i_4 zbog pravila $n_5 \leftarrow i_4$.

0	1	2	3	4	5	6	7	8	9
A	i_3	b	y	$\{h\}$	n_3	n_4	i_4	b	n_5