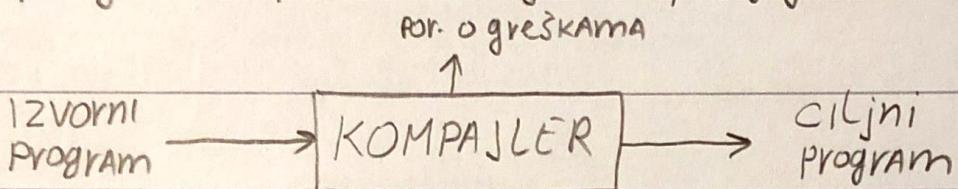


Programski

Prevodioci

UVOD

- * ZADATAK PP je DA prenove programe pisane izvornim prog.jez u programe pisane ciljnim prog.jez



PROG.JEZ. VISOKOG NIWOA



* Prevodenje → prepoznavanje ISKAZA (rečenica) IZVORNOG JEZ., SASTOJI SE OD prepoznavanja reči (simbola) i rečenice (pojma)

Prevodenje = ANALIZA + SINTEZA + OPTIMIZACIJA

ANALIZA - prepoznavanje ISKAZA IZVORNOG prog.

SINTEZA - generisanje ISKAZA ciljnog prog.

OPTIMIZACIJA - poboljšanje prog.

*Analiza

(SKENER)

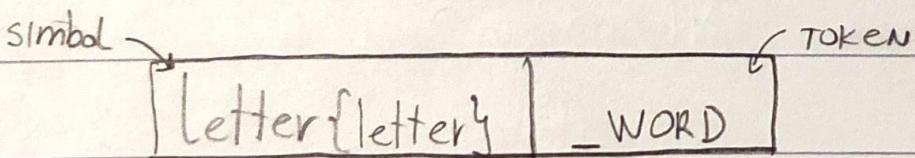
- ① Leksička - prepoznavanje reči (simbola), OTKRIVANje pogrešnih simbola
(PARSER)
- ② Sintakstika - prepoznavanje ISKAZA (rečenica), OTKRIVANje formalno pogrešnih ISKAZA
- ③ Semanticka - prepoznavanje ZNAČENJA ISKAZA, OTKRIVANje semantički pogrešnih ISKAZA

(GENERATOR KODA)

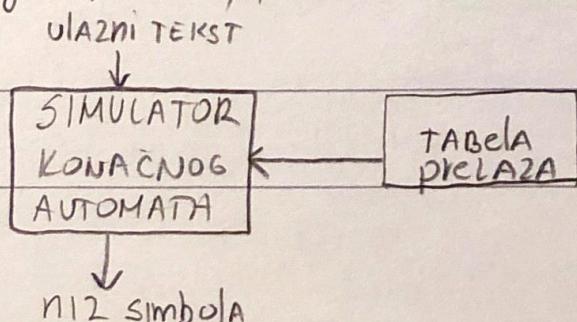
- *Sinteza - prethodno prepoznati ISKAZI (rečenice) izvornog prog. jez se prevode na rečenice ciljnog jez. (Assemblerski)

Skener

- * Prepozna je leksičke jed. (simbole) iz ulaznog teksta.
- * Preuzima znak po znaku, pokušava da prepozna simbol, ignorise znakove koji razdvajaju simbole (" ", "\t", "\n") i reaguje na nedozvoljene znakove.
- * String simbola = leksema
- * Klasa simbola (skup stringova) = TOKEN
- * Identifikator: a, Aa, aA, A2, BA..
- * Broj: 0, 125, 1250...
- * Ključna reč: "if", "else", "for", "while"
- * belina: sekvenca praznih mesta. (" \n \t" → jedan token beline)



- * Deli ulazni tekst na lekseme, identificuje token svake lekseme, tako što preuzeće opis simbola i na osnovu njega prepozna i klasificuje simbole
- * Dijagram prelaza je usmereni graf u čiji sastav ulaze čvorovi (stanja) usmerene spojnice (mogući prelasci), labele i znakovi (obrazuju labele)



* Regularni izrazi

* $0|1|N$ puta $b|a$ $\rightarrow bba, baa\dots$

+ $1|N$ puta $b|a$ $\rightarrow ba, baa\dots$

? $0|1$ put $b|a$ $\rightarrow b, ba$

| Alternative $a|b$ $\rightarrow a \text{ ili } b$

() grupisanje izraza

[] NAVODENJE KLASI ZNAKOVA $[0-9]$ $\rightarrow 0, 1\dots 9$

• Bilo koji znak sem " $\backslash n$ "

\ Poseban znak se tretira kao običan \. $\rightarrow \cdot$

{ } Def. br. ponavljanja $a\{2-4\}$ $\rightarrow aa, aaa, aaaaa$

^ Ne može taj znak H \rightarrow SVI SEM "H"

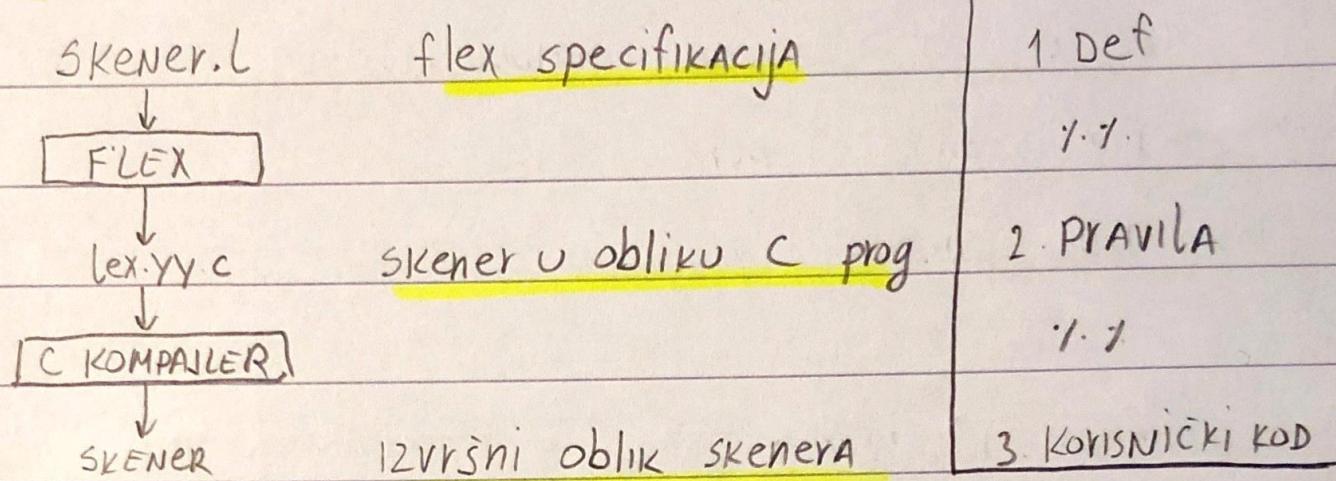
" " BAŠ TAJ ZNAK

* Generator skenera

pomoću regularnih izraza se mogu opisati simboli,

Generator tabele prelaza određuje ponašanje skenera (FLEX)

* Flex



- Flex generiše skener u obliku C fje yylex() koja se pravi na osnovu reg. izraza i njima pridruženim akcijama. Flex zahteva def fje yywrap() koja opisuje ponašanje skenera kada HADE NA EOF znak.

- yylex() ponavlja prepoznavanje simbola i izvršava zadate akcije sve dok se u okviru akcija ne izvrši return iskaž

- Globalne promenljive

- yytext → pristup stringu poslednjeg prepoznatog simbola
- yylen → dužina stringa poslednjeg prepoznatog simbola
- yylineno → sadrži broj trenutne linije
- yyval → prenos vrednosti prepoznatog simbola

Parser

- * Proverava da li je ulazni niz simbola (tokens) u skladu sa gramatikom
- ulaz → niz tokens od skenera
- niz → stablo parsiranja programa

niz
KARAKTERA → SKENER → niz
TOKENA → PARSER → STABLO
PARSIRANJA

- * Da bi razlikovalo ispravne od nespravnih nizova tokens potrebno mu je
 - JEZIK ZA OPIS ISPRAVNIH NIZOVA TOKENA
 - METODA ZA RAZLIKOVANJE ISPRAVNIH OD NESPRAVNIH NIZOVA TOKENA.
- * BNF gramatika se sastoji od pravila koja određuju dozvoljene načine redanja pojmove i simbola
- * JEDAN OD POJMOVE JE polazni pojam (koren STABLA)
- * Pojmovi → SINTAKSNA ANALIZA
- * Simboli → LEXIČKA ANALIZA
- * EBNF (Extended BNF)-na desnoj str pravila može:
 - [...] → sadržaj zagrada može ^{se} pojaviti jednom ili ni jednom (0 ili 1)
 - {...} → sadržaj zagrada može ~~se~~ pojaviti ni jednom, jednom ili više puta (0+1)
 - (...) → grupisanje
 - | → Alternative (ili)

- * Pravila se koriste u parseru tako što se leva str menja desnom
- * Simboli se ne mogu dalje zamjeniti, ne postoje pravila za simbole
- * Izvođenje je niz promenljivih pravila, počinje se od početnog pojma, primjenjuje se zamena pravila, jedan po jedan pojam, stablo gde su listovi simboli, a čvorovi pojmovi
- * Jedno stablo parsiranja može imati više izvođenja
- * Gramatika je duosmislena kada za jedan ulazni string ima više stabala - lošo
- * Bison

→ Vrednost pojma sa leve strane pravila

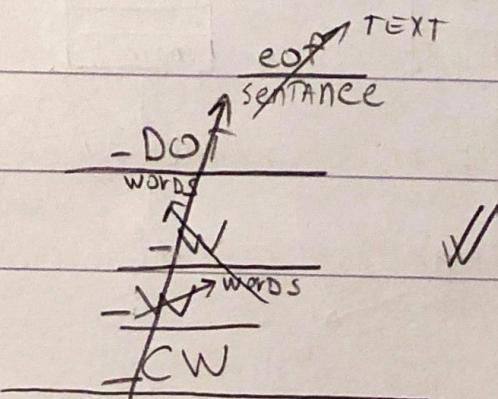
\$i → Vrednost pojma ili simbola na desnoj strani pravila

NA steku \$i označava lokacije ispod vrha steka, ## na koncu njihove redukcije označava vrh steka

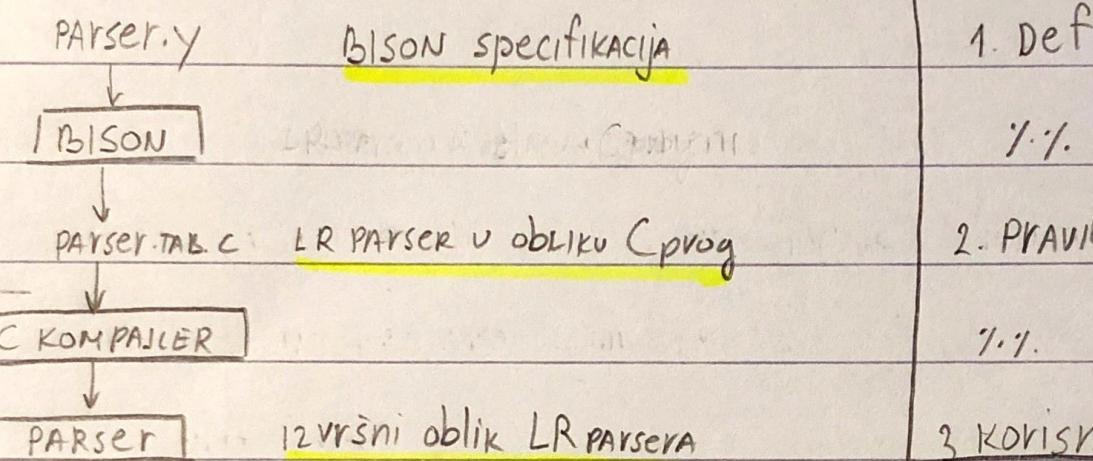
* Parser pokušava da prepozna desnu stranu pravila i da je redukuje u levu stranu, sve dok ne dođe do polarnog pojma (silazno parsiranje)

* "Ovo je text."

- CW - w - w - DOT EOF



* BISON



BISON specifikacija

1. Def

YY.

2. PRAVILA

YY.

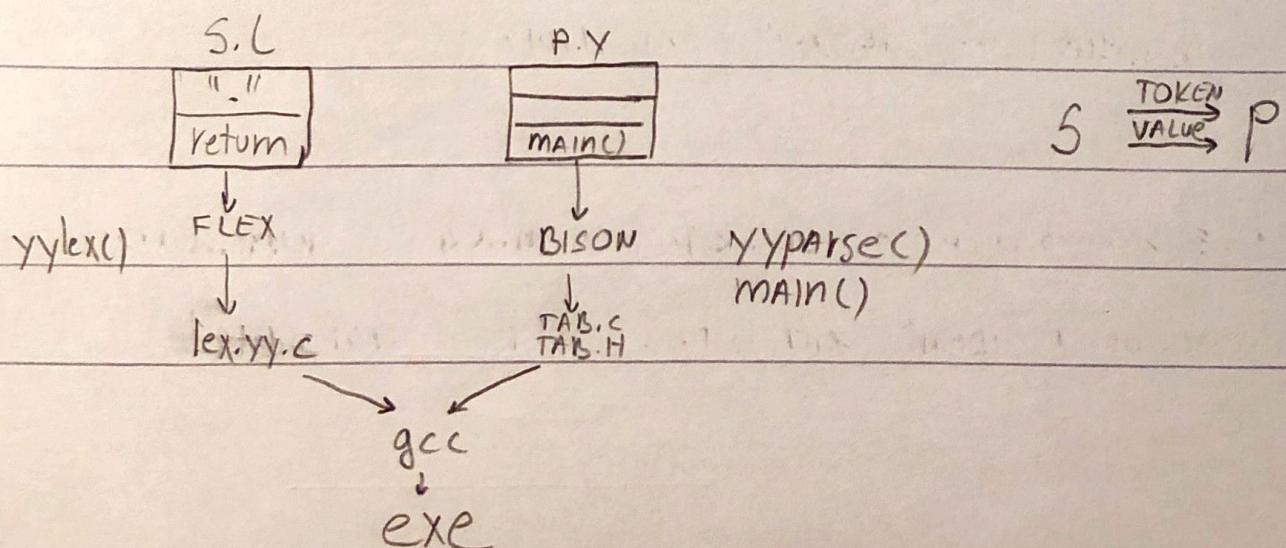
3 KORISNIČKI KOD

- Generiše LR parser u obliku C fje `YYPARSE()` koja ukazuje na grešku ako vrati razlicito od 0
- Pravila u BNF notaciji
- Podrazumeva da svaki pojam i simbol iz pravila poseduju vrednost, vrednost simbola odreduje skener, a pojava parser.

$$A \rightarrow B \{ \$\$ = 5; \} \quad C \quad \{ \$\$ = 12; \} \Rightarrow A = 5$$

SAMO NE NA PRIMESNO
DESNE STRANE PRAVILA

$$A \rightarrow B \{ \$\$ = 5; \} \quad C \quad \{ \$\$ = 1; \} \Rightarrow A = 1$$



Mini C

* Simboli

- Identifikator(ime): Identifikator → slovo (slovo | cifra)*
- celobrojni označeni literal(konstanta): int_literal → cifra +
- celobrojni neoznačeni literal(konstanta): uint_literal → cifra + ("U" | "U")

* Pojmovi

funkcija → type - identifikator "(" - parametar ")" telo
tip → "int" | "unsigned"

parametar → | tip - identifikator

telo → "{" lista_varijabli lista_stejmenta "}"
| "{" lista_stejmenta "}"

lista_varijabli → | Lista_varijabli varijabla

varijabla → tip - identifikator ";"

lista_stejmenta → lis | Lista_stejmenta stejment

stejment → dodela | if | return | višestruki

dodata → - identifikator "=" num-exp ";"

num-exp → exp | num-exp "+" num-exp | num-exp "-" num-exp

exp → literal | - identifikator | funkcija | "(" num-exp ")"

funkcija → - identifikator "(" Argument ")"

Argument → | num-exp

_if → "if" "(" rel-exp ")" stejment

| "if" "(" rel-exp ")" stejment "else" stejment

rel-exp → num-exp - OZNAKA num-exp

OZNAKA = <, <=, >, >=, ==, !=

_return → "return" num-exp ";"

visestruki → "{" Lista-stejmenta "}"

* Semantika

- STANDARDNI IDENTIFIKATORI SU REZERVISANE REC'I, IDENTIFIKATOR MAIN

- OPSEG VIDLJIVOSTI IDENTIFIKATORA

1. Globalni → Def na nivou programa, važe do kraja prog.

STACIONI U MCM.
LOKACIJAMA

2. Lokalni → Def u okviru fje, važe do kraja tела gde su def

DINAMIČNI NA
STEKU

- JEDNOZNAČNOST IDENTIFIKATORA?

- LEVA I DESNA STR DODELE MORAJU BITI ISTOG TIPOA

- PROVERA globalnih i lokalnih identifikatora → PODRUČJE VAŽENJA

* Identični globalni i lokalni identifikatori mogu postojati !!!

* Identični lokalni identifikatori u različitim f'jama mogu postojati !!!

Semantika

- * Opisuje se na neformalan način
- * Identifikatori su:
 - Rezervisane reči - Int, Unsigned, if, else, return
 - MAIN - PODRAZUMEVA SE DA POSTOJI, IZVRŠAVANJE POČINJE OD NJE
- * Globalni identifikatori su def na nivou programa (imenafje) i važi od mesta definicije do kraja prog. teksta. smeštaju se u mem. lokacije
- * Lokalni identifikatori su def. u okviru fja (VARI PAR) i važe od od mesta def. do kraja tela fje u kojoj su def. SVAKA fja poseduje svoje lok. promenljive.
 - NA STEKU
- * Ne može DVA ISTA glob. ili lok. identifikatora, ali može isti glob i lok.
- * Ako postoje DVA ISTA identifikatora, jedan glob drugi lok, van te fje VAŽI globalni A u telu fje lokalni.
- * Identifikator main je rezervisan SAMO NA globalnom nivou, može se koristiti na lokalnom. To ne važi za rezervisane reči.
- * Deo steka koji se ZAUZIMA ZA IZVRŠAVANJE neke fje se zove frejm
- * registr %14 je pokazivač frejma, parametri se adresiraju preko njega, +
8(%14), 12(%14)... . lokalne promenljive se adresiraju takođe preko %14, -4(%14), -8(%14)...

- * Na leve strane ISKAZA dodele imenima mora biti deklarisano pre upotrebe i mora biti ili lok. promenljiva ili parametar fje
- * Tipovi sa leve i desne strane ISKAZA dodele moraju biti isti
- * Tipovi operanada aritmetičke operacije moraju biti isti.
- * Ime mora biti deklarisano pre upotrebe
- * Vrednost pojma exp je indeks u tabeli simbola gde se nalazi rez izraza!!!
- * Vrednost pojma literal je indeks u tabeli simbola na kom se on nalazi.
- * Tip izraza iz RETURN ISKAZA fje mora biti istog tipa sa tipom fje
- * Tipovi operanada relacionih op. moraju biti isti.
- * Argumenti poziva fje moraju da se slazu po broju ^{i tipu} SA parametrima fje
- * Povratna vrednost fje se nalazi u registru %.13
- * Vrednost pojma function-call je reg %.13

* Tabela simbola

- pronađeni globalni i lokalni identifikatori (simboli) se čuvaju tu, zajedno sa odg. atributima identifikatora
- operacije su ADD, delete, find, set, get
- Tabela je u stvari hash tabela, zbog find
- Za svaki identifikator postoji oznaka njegovog tipa
- Struktura
 - Name → string identifikatora
 - Kind → vrsta simbola (enumeracija kinds)
 - Type → tip simbola (enumeracija types)
 - Atr1 → Atribut, razlike vrednosti za razlike vrste simbola
 - Atr2 → za identifikator fje, tip Argumenta
- U tabeli mogu biti smestene konstante i oznake radnih registara

Primeri PP

① Sintaksnu grešku predstavlja?

- Pogrešan redosled simbola

② Pretraga tabele simbola vrši se pomoću fje?

- `lookup-symbol()`

③ Koliko ima globalnih identifikatora?

`int f(int p) {`

`int m;`

`return m;` \Rightarrow 2 globalna identifikatora

}

`int main() {`

}

④ Kom delu miniC koda odg Asemblerski kod?

`SUBS -8(%14), -4(%14), %0`

`ADDS -4(%14), -4(%14), %1`

`SUBS %0, %1, %0`

`MOV %0, -4(%14)`

- $a = (b-a) - (a+a)$

⑤ Najviše kompjlera radi

- lokalnu optimizaciju

⑥ Koji string odg. reg. izrazu $0[xX][0-9a-fA-F]^{\{1,4\}}$

- $0x1234$ i $0x1010$

⑦ Koliko puta skener prepozna token identifikatora (-ID)

int main() {

$a = x * 5;$

$\Rightarrow 3$ tokena - ID

}

⑧ Koliko ima lokalnih identifikatora

int main() {

int x; int y; $\Rightarrow 2$ lokalna identifikatora

return x+y;

}

⑨ Koji pojmovi se prepoznaju prilikom parsiranja $x=y+5;$

- exp i num-exp

⑩ $a: A \{ \} B \{ \}$ #3 odgovara kome?

- B

⑪ KAKVA greska postoji u miniC kodu?

int x; int y;

X = X + Y

- SINTAKSNA (;)

⑫ Upisati vrednost polja attr1 u tabelu simbola za f

int f() {

 int b; ⇒ 2 (DVA kom)

 return b + 2 * f();

}

⑬ Koji niz tokena će biti prepoznat za $a = f();$

- -ID, -ASSIGN, -ID, -LPAREN, RPAREN, -SEMICOLON

⑭ Donja crta, jedno do dva mala i 0,1 ili N velikih i cifara?

- ""[a-z][a-z]?[A-Z0-9]*" ili ""[a-zA-Z]{1,2}[A-Z0-9]*"

⑮ Koji pojmovi se prepoznaaju u $x = x + a;$

- Literal i exp

⑯ Prilikom generisanja koda, najčešće korisćene vr. se smestaju u?

- Registre

⑰ Optimizacija koja se vrši na jednom baznom bloku je?

- LOKALNA

⑯ Optimizacija petlje ima za cilj?

- manji broj naredbi u telu petlje

⑰ Šta se u toku opt. međukoda radi sa suvišnim naredbama?

- izbacuju se

⑯ Kom delu koda odgovara?

SUBS \$2, -8(%14), %0

MOV %0, -4(%14)

- $A = 2 - B;$

⑯ Formalna gramatika se sastoji od?

- pojmove i simbola

⑯ Interpreter sadrži?

- sintaksnu analizu i generisanje međukoda

⑯ U hipotetskom Asemblerском jez. pok. frejma je?

- %14

⑯ Koliko ukupno ima reg hipotetski AsemblerSKI jez?

- 16

⑯ $a = b + (c - d);$

- ID, - ASSIGN, - ID, - AROP, - LPAREN, - ID, - AROP, - ID, RPAREN,

- Semicolon

⑥ Attr za b u

int f() {

int a; int b; ... \rightarrow Attr = 2

return a+b+5;

}

⑦ KAKVA je grešKA za

int f (int p) {

}

int main () { \Rightarrow SemantičKA

return f();

}

⑧ Poželjne osobine kompjajlera?

- Dobra dijagnostika grešaka i dobra optimizacija

⑨ Za lokalne identifikatore važi:

- Postoje samo za vreme izvršavanja fje

- Oslobađaju se na kraju fje

⑩ Koji reg izraz def string od min 2 znaka?

- [a-zA-Z][0-9]?[A-Za-z]+

61) Formalna gramatika se sastoji od?

- Startnog pojma i Simbola.

62) Koja greška postoji u kodu $a = x + y ? ;$

- Leksička

63) Koji parseri su pogodni za automatsko generisanje?

- LR parseri.

64) Koja vrsta greške?

int f (int num){

 return num * num \Rightarrow Leksička ("*")

}

65) Da li je kod ispravan?

int f () { }

int main () {

 int n;

 f = n + 5; \Rightarrow Ne (f ja sa leve str)

}

56) Koja vrsta greske?

*RETURN m → isto

int main() {

 int a; unsigned b;

 f(a>b){ }

⇒ SEMANTICKA

}

57) Vrednost Attr od m?

int main(){

 int n; unsigned m;

⇒ 2 (druga varijabla po redu)

n=6

}

58) Koja vrednost u Attr od f

int f(int p){ } ⇒ 1 (jedan parametar)

59) Koja vrednost u ptype (5 kolona) od f

int f(unsigned p){ } ⇒ UINT (unsigned p)

60) Semantika formalnog jez se opisuje?

- Neformalno

61) Semanticka provera u function-call?

- Uporediti br parametara i Arg i tipove parametara i Arg.

Greške:

Leksička \Rightarrow pogrešni simboli (cout, nt)

Sintaksna \Rightarrow pogrešni ISKAZI (fale "(, ")", ";" ...)

Semantička \Rightarrow Nedefinisane promenljive (int a; b = 3;)

Mini C - Semantika

* Tabela simbola je struktura podataka koja čUVA identifikatore i sve inf o identifikatorima

→ Name - String identifikatora

→ Kind - vrsta simbola

→ type - tip simbola

→ Atr1 - redni br. lok.prom, parametra; br par fje

→ Atr2 - tip parametra za fju

* Funkcije tabele

→ Int insert_symbol - ubacuje simbol u tabelu i vracá njegov index u njoj.

→ Int insert_literal - ubacuje literal u tabelu i vracá njegov index u njoj.

→ Int lookup_symbol - traži simbol u tabeli i vracá njegov index u njoj

→ Int lookup_literal - traži literal u tabeli i vracá njegov index u njoj

→ set i get - izmena i čitanje određenih vrednosti iz tabele

→ Void clear-symbols - briše deo tabele, od prosledjenog indeksa pa do kraja

→ Void print_symtab - prikaz svih popunjениh el. tabele

→ Void init_symtab - inicijalizacija tabele simbola

→ Void clear_symtab - brisanje svih el. tabele

- * Semantickе provere se implementiraju pomoću Tabele simbola.
- * U tabeli simbola mogu biti smještene oznake radnih reg., smještju se za vreme inicijalizacije $\%0 \rightarrow \text{idx } 0$, $\%12 \rightarrow \text{idx } 12 \dots$
- * Br el. u tabeli ograničava br. identifikatora u programu
- * Br el. niza sa tipovima parametara ograničava br. parametara fje.
- * Br radnih reg ograničava rad kompjlera

Dvosmislena gramatika

- * Gramatika je dvosmislena kada za jedan ulazni string postoji više stabala parsiranja ili više od jednog izvođenja.
- * Problem dvosmislenosti gramatike je nudio da se dve ili više ravnopravnih mogućnosti izvođenja, time je ostavljeno kompjuteru da se odluci za jednu od više interpretacija programa.
- * Rešenje je deklaracija leve asocijativnosti %left +,*
- * Generatoru LR parsera moraju biti saopštene nedvosmislene gramatike da bi on u svaki element tabele Akcija i prelaza mogao da smesti označku samo jedne akcije, u suprotnom se može javiti više akcija kao kandidata za isti el. tabelu → konflikt
- * Bison rešava konflikte na dva načina, kod shift-reduce prednost daje shift akciji, kod reduce-reduce daje prednost reduce naredbi
- * Deklaracije prioriteta mogu da se definišu samo jednom za jedan token
- * %prec modifikator deklariše prioritet nekog pravila

RUKOVANJE GREŠKAMA

- * NAKON OTKRIVANJA GRESKE U PROG. TEKSTU POTREBNO JE NAVESTI OPIS GRESKE I UKAZATI NA NJENO MESTO
- * NA MESTO POJAVE GRESKE UKAZUJU BR LINIJE, POGREŠNI DEO TEKSTA.
- * Ukoliko se pojavi yyerror greska, kompjajler će prijaviti gresku i nastaviti kompjajliranje
- * Token error hvata bilo koju gresku postavljenu između DVA TOKENA tj. simbola

Mini C kompjajler

- * Prevodi sa jezika miniC na hipotetski Asemblerски jez.
- * Prijavljuje greške u izvornom miniC programu
- * Leksička analiza (skener) - deli ulazni string na simbole prog. jez.
- * Sintaknska analiza (parser) - provera ispravnog redosleda simbola
- * Semantička analiza (parser) - proverava konzistentnost prog.
- * Generisanje koda (parser) - prevodi prog. na ciljni (hip. ASM) jezik
- * Tabela simbola:
 - Koriste je sve faze kompjajiranja
 - Struktura podataka u kojoj se čuvaju sve inf. o svim simbolima
 - Na osnovu tabele moguća je semantička analiza i generisanje koda.
- * Kompajler treba da saopšti grešku i da se oporavi od iste radi detektovanja nove.
- * Ukoliko bi se kompjajleru prosledila datoteka sa neispravnim programom on bi prijavio tačnu lok. za implementiranu ili samo da postoji greška u slučaju neimplementirane greske.

- * CMPx ima dva ulazna operanda, postavlja bit status reg.
- * Naredba bezuslovnog skoka smješta vrednost ulaznog operanda u programski brojač
- * Uсловni skok ima ulazni operand
- * Ispunjenošć условia zavisi od bita status reg.
- * CALL stavlja trenutnu vrednost prog. br na vrh steka a ulazni op. u prog. br.
- * RET preuzima vrednost sa vrha steka i smješta u prog. br.
- * Izuzetak u ADDx, SUBx... kad rez ne može da stane u ulazni op.
- * WORD je zauzimanje mem. lok, "broj" je broj učestalih lokacija
- * Zauzimanje reg se radi ADD/SUB, oslobađanje sa MOV
- * Fja čuva lok. promenljive i parametre na stek frejmu
- * Brojač lok. promenljivih je VAR-NUM
- * Ako fja ne sadrži return, povratna vrednost je vrednost zatečena u reg 13
- * Konstante i promenljive direktno mogu na stek u slučaju poziva fje

- * Indeks registra = Indeks u tabeli simbola
- * Inicijalno punjenje TABele simbola sa radnim reg
- * take-reg, free-reg, free-if-reg
- * gen-sslab, gen-snlab,
- * gen-mov (int ulazni-ind, int izlazni-ind)
- * gen-Arith (int stmt, int op1-ind, int op2-ind)
- * Posle prepoznavanja parametara nema gen KODA, samo semantičke akcije