Gramatika mC programskog jezika

- Gramatika za programski jezik `m` C
  - podskup programskog jezika C
  - izražena u BNF notaciji
  - analiza svih simbola mC gramatike

Skener za mC

- Za rezervisane reči (**`if`**, **`return`**, ...) je dovoljno da parser dobije token
- Za relacione operatore je potrebno da parser dobije token i vrstu operatora koji predstavlja vrednost simbola
- Za aritmetičke operatore dovoljno je da parser dobije token i vrstu operatora
- Za numeričke konstante je potrebno da parser dobije token i pokazivač stringa konstante koji predstavlja vrednost simbola
- Za identifikatore je potrebno da parser dobije token i pokazivač stringa identifikatora koji predstavlja vrednost simbola

Skener za mC

```
%{
   union {
      int i;
      char *s;
   } yylval;

   enum { _TYPE = 1, _IF, _ELSE, _RETURN, _LPAREN, _RPAREN,
          _LBRACKET, _RBRACKET, _SEMICOLON, _ASSIGN, _AROP,
          _RELOP, _ID, _INT_NUMBER };
   char *token_strings[] = { "NONE", "_TYPE", "_IF", "_ELSE",
         "_RETURN", "_LPAREN", "_RPAREN", "_LBRACKET",
         "_RBRACKET", "_SEMICOLON", "_ASSIGN", "_AROP",
         "_RELOP", "_ID", "_INT_NUMBER" };
   enum { ADD, SUB, EQ, NE };
   char *value_strings[] = { "ADD", "SUB", "EQ", "NE" };
%}
%%
```

Skener za mC

```
[ \t\n]+                { /* skip */     }

"int"                   { return _TYPE; }
"if"                    { return _IF; }
"else"                  { return _ELSE; }
"return"                { return _RETURN; }

"("                     { return _LPAREN; }
")"                     { return _RPAREN; }
"{"                     { return _LBRACKET; }
"}"                     { return _RBRACKET; }
";"                     { return _SEMICOLON; }
"="                     { return _ASSIGN; }
"+"                     { yylval.i = ADD; return _AROP; }
"-"                     { yylval.i = SUB; return _AROP; }
"=="                    { yylval.i = EQ; return _RELOP; }
"!="                    { yylval.i = NE; return _RELOP; }
```

Skener za mC

```
[a-zA-Z][a-zA-Z0-9]* { yylval.s = yytext; return _ID; }
[+-]?[0-9]{1,10}     { yylval.s = yytext; return _INT_NUMBER;}

\/\/.*               { /* skip */ }
.                    { printf("line %d: LEXICAL ERROR on char"
                              "%c\n", yylineno , yytext[0]);}
%%
int main() {
   int tok;
   while(tok = yylex()) {
      printf("%s", token_strings[tok]);
      if(tok == _ID || tok == _INT_NUMBER)
         printf(": %s\n", yylval.s);
      else
         if(tok == _AROP || tok == _RELOP)
            printf(": %s\n", value_strings[yylval.i]);
         else printf("\n");
   }
}
```

Skener – primer izlaza skenera

```
int boolval() {
   int x;
   if(x != 0)
      return 1;
   else
      return 0;
}
```

```
_TYPE
_ID: boolval
_LPAREN
_RPAREN
_LBRACKET
_TYPE
_ID: x
_SEMICOLON
_IF
_LPAREN
_ID: x
_RELOP: NE
_INT_NUMBER: 0
_RPAREN
_RETURN
_INT_NUMBER: 1
_SEMICOLON
_ELSE
_RETURN
_INT_NUMBER: 0
_SEMICOLON
_RBRACKET
```

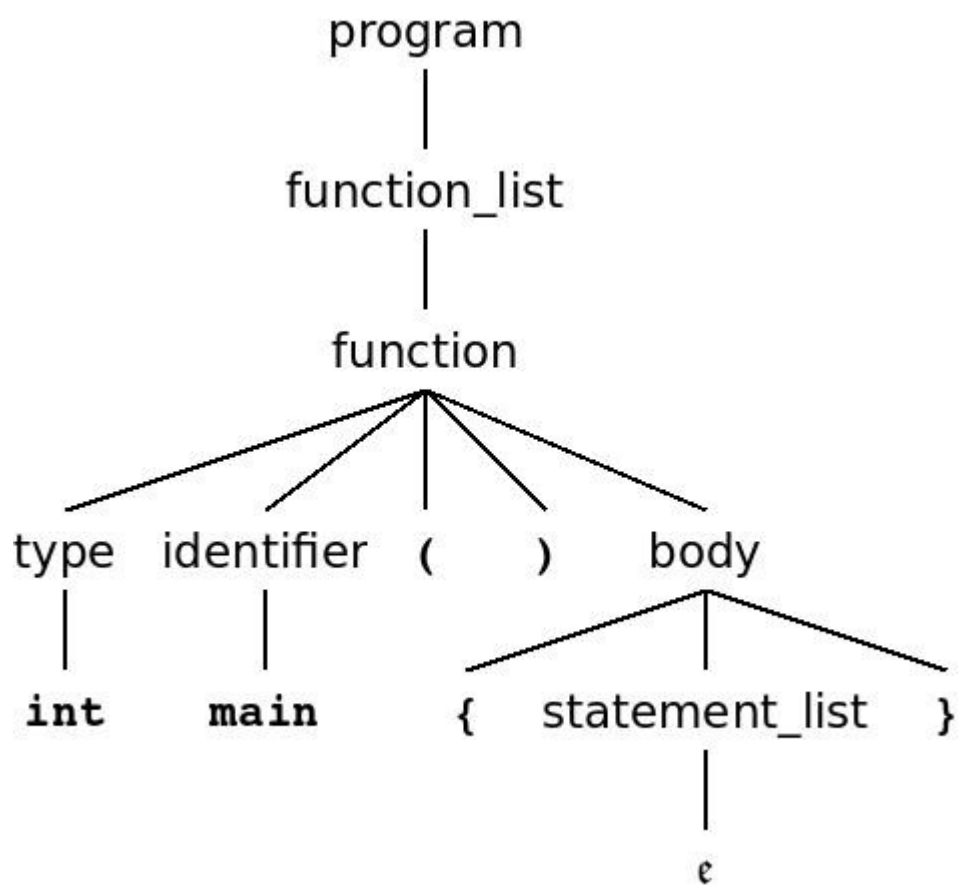
Parser za mC gramatiku

- Niz izvođenja s leva za iskaz: **`int main() {}`**

program =>
  function_list =>
  function =>
  type identifier **( )** *body* =>
  int identifier **( )** *body* =>
  int main ( ) *body* =>
  int main ( ) { *statement_list* **}** =>
  int main ( ) { }

Parser za mC gramatiku

- Stablo parsiranja za iskaz: **`int main() {}`**



Parser - dvosmislene gramatike

- Primer dvosmislene gramatike za iskaz

```
if (a != b) if (a == c) b = a; else b = c;
```

```
if (a != b)
   if (a == c)
      b = a;
else
   b = c;
```

```
if (a != b)
   if (a == c)
      b = a;
   else
      b = c;
```

Parser - dvosmislene gramatike

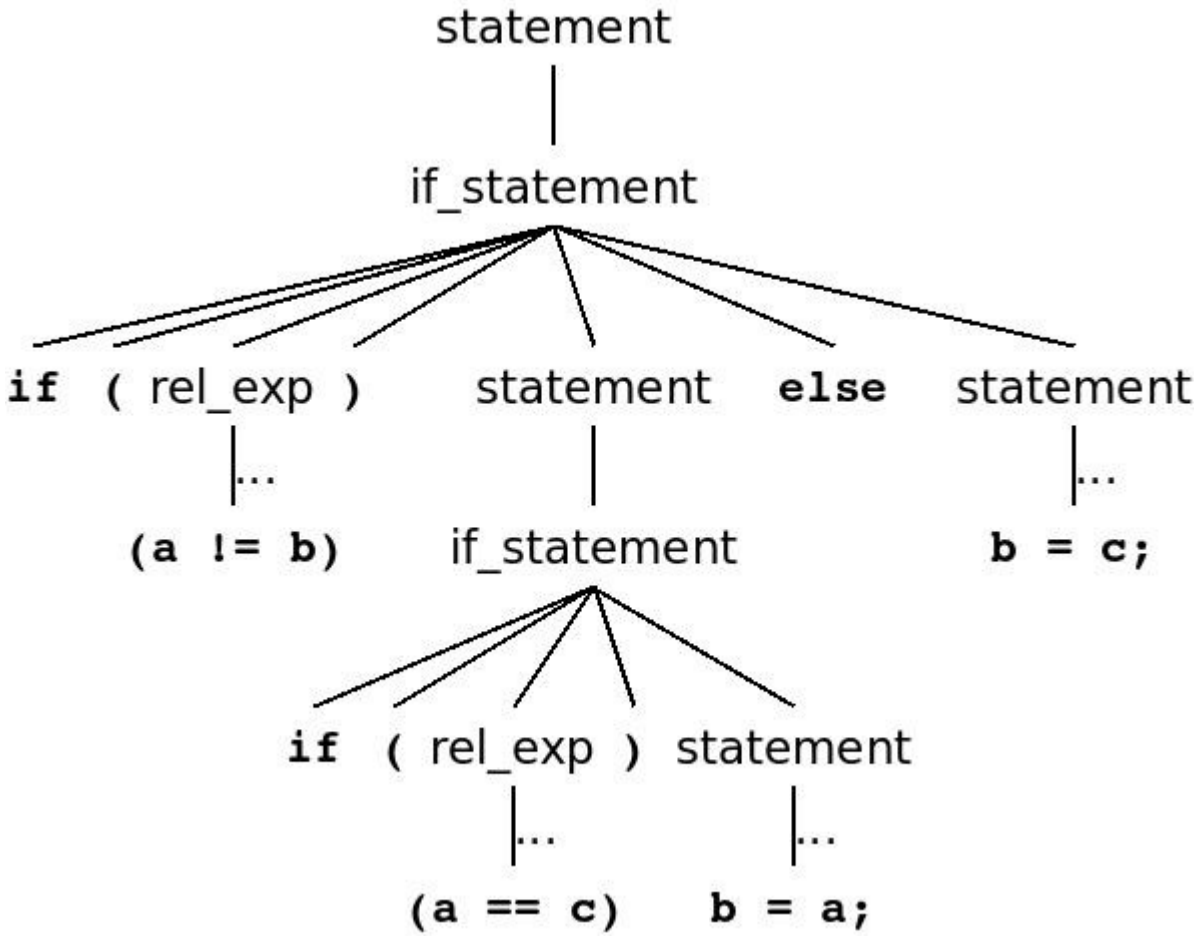
`if (a != b) if (a == c) b = a; else b = c;`

- netačno izvođenje s leva

statement =>
if_statement =>
if (*rel_exp***)** *statement* **else** *statement* =>*
if (a != b) *statement* **else** *statement* =>*
if (a != b) *if_statement* **else** *statement* =>
if (a != b) if (*rel_exp***)** *statement* **else** *statement* =>*
if (a != b) if (a == c) *statement* **else** *statement* =>*
if (a != b) if (a == c) b = a; else b = c;

Parser - dvosmislene gramatike

- Stablo parsiranja za netačno izvođenje s leva



Parser - dvosmislene gramatike

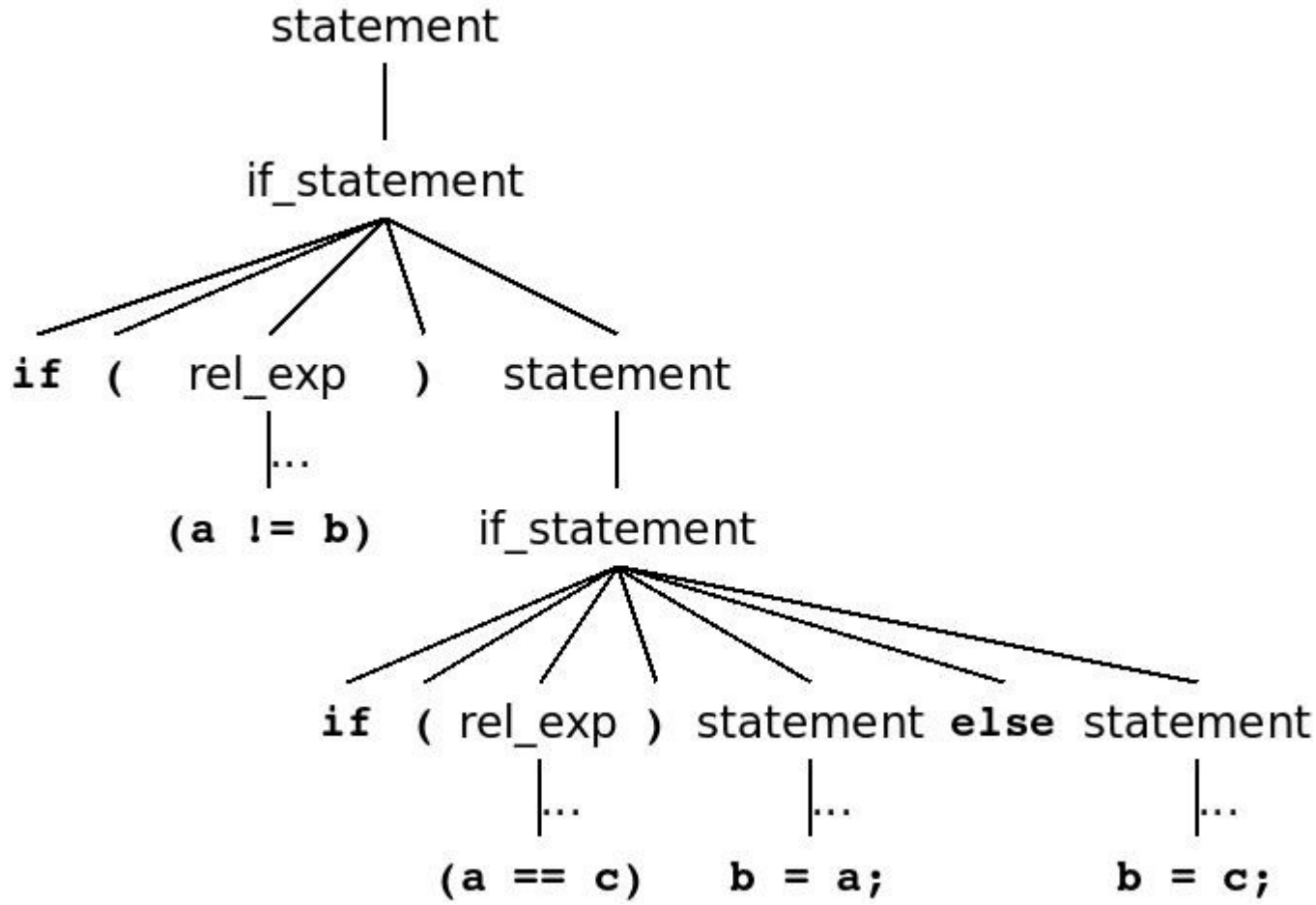
`if (a != b) if (a == c) b = a; else b = c;`

- tačno izvođenje s leva

statement =>
if_statement =>
if (*rel_exp***)** *statement* =>*
if (a != b) *statement* =>
if (a != b) *if_statement* =>
if (a != b) if (*rel_exp***)** *statement* **else** *statement* =>*
if (a != b) if (a == c) *statement* **else** *statement* =>*
if (a != b) if (a == c) b = a; else b = c;

Parser - dvosmislene gramatike

- Stablo parsiranja za tačno izvođenje s leva



Parser - dvosmislene gramatike

- Problem dvosmislenosti gramatika je što one u pojedinim koracima izvođenja nude dve ili više ravnopravnih mogućnosti
  - tako dovode parser u situaciju da ne može automatski da se jednoznačno opredeli za jednu mogućnost
  - posledica je da razni parseri mogu različito interpretirati isti program

Parser - dvosmislene gramatike

- Ukoliko je stanje na steku sledeće:

  `if rel_exp1 if rel_exp2 statement1`

  i ukoliko parser kao sledeći token dobija **`else`**, može se obaviti neispravna redukcija (*reduce*) nad poslednja 3 elementa:

  `if rel_exp1 statement [else statement 2]`

  nakon čega će se **`else`** i iskaz iza njega priključiti prvoj **`if`** naredbi, ili se može obaviti ispravno šiftovanje (*shift*):

  `if rel_exp1 if rel_exp2 statement1 else [statement2]`

  nakon čega će se **`else`** i iskaz iza njega priključiti drugoj **`if`** naredbi

Parser - Bison

- Generatoru LR parsera moraju biti saopštene nedvosmislene (*unambiguous*) gramatike da bi on u svaki element tabele akcija i prelaza mogao da smesti oznaku samo jedne akcije
  - u suprotnom se može javiti više akcija kao kandidata za isti element tabele
  - ovakve situacije se nazivaju **konflikti**
  - konflikti se razrešavaju davanjem prednosti jednoj akciji

Parser - Bison

- Kako Bison razrešava konflikte:
  - kod *shift-reduce* konflikta prednost daje *shift* akciji
    - to rešava problem jednoznačne interpretacije **`if else`** iskaza
  - kod *reduce-reduce* konflikta prednost daje *reduce* akciji po pravilu koje je ranije navedeno
- Ako prethodni podrazumevajući način za razrešavanje konflikta, koji primenjuje generator, nije prihvatljiv za korisnika, potrebno je da on generatoru saopšti na koji način konflikt treba razrešiti
  - navođenjem prioriteta (*precedence*) operatora i redosleda njihove primene (*associativity*)
  - izmenom gramatike *****

Parser - Bison

- *reduce-reduce* konflikt se javlja ukoliko postoji 2 ili više pravila koja se mogu primeniti na isti niz ulaznih simbola. Ovakva situacija označava ozbiljnu grešku u gramatici.

```
statement_list
    :  /* empty */
          { printf("empty statement_list \n"); }
    |  maybe_statement
          { printf("maybe statement_list \n"); }
    |  statement_list statement
          { printf("added statement \n"); }
    ;
maybe_statement
    :  /* empty */
          { printf("empty maybe_statement\n"); }
    |  statement
          { printf("single statement \n"); }
    ;
```

Parser - Bison

- Greška je u dvosmislenosti: postoji više od jednog načina da parser od pojma **`statement`** dođe do pojma **`statement_list`**.
  - prvi način je redukcija pojma **`statement`** u pojam **`maybe_statement`**, a zatim redukcija u pojam **`statement_list`** preko drugog pravila
  - drugi način je da se **ε** redukuje u **`statement_list`** preko prvog pravila, a zatim da se **`statement_list`** kombinuje sa **`statement`** pomoću trećeg pravila.
- Takođe, postoji više od jednog načina da se **ε** redukuje u **`statement_list`**. Redukcija je moguća
  - direktno preko prvog pravila,
  - ili indirektno preko pojma **`maybe_statement`** i drugog pravila.
- Ove razlike utiču na to koje akcije će biti izvršene. Jedan redosled parsiranja će izvršiti akciju uz drugo pravilo, a drugi akciju uz prvo pravilo i zatim akciju uz treće pravilo. U ovom primeru, menja se izlaz programa.

Parser - Bison

- Bison razrešava *reduce-reduce* konflikt tako što odabira pravilo koje je ranije navedeno u gramatici, ali vrlo je rizično osloniti se na ovo. Svaki *reduce-reduce* konflikt se mora dobro proučiti i eliminisati.
- Ispravan način da se definiše pojam **`statement_list`** je:

```
statement_list
    :  /* empty */
          { printf("empty statement_list \n"); }
    |  statement_list statement
          { printf("added statement \n"); }
    ;
```

Parser - Bison

- Treba napomenuti da će zbog načina nakoji Bison radi (zbog toga što se posmatra samo jedan token unapred), sledeća pravila dovesti do konflikta:

```
X : A B C {akcija1} D E F
  | A B C {akcija2} G H I
  ;
```

- Kod ovakvih pravila, akcije se moraju pomeriti, tako da se pre njih nalazi jednoznačan niz simbola
  - u primeru, ako se akcija pomeri iza D, odnosno G, konflikta neće biti

Parser

- Prepoznavanje ispravnog niza simbola izvornog programa odgovara konstrukciji stabla parsiranja
- To je dovoljan preduslov za sintezu ciljnog programa, jer nakon prepoznavanja iskaza izvornog programskog jezika postaje moguća njihova zamena iskazima ciljnog programskog jezika

- primer – mC parser

Parser - postupak sa greškama

- Nakon otkrivanja greške u programskom tekstu potrebno je navesti
  - opis greške i
  - ukazati na mesto njene pojave
- Na mesto pojave greške ukazuju
  - broj linije programskog teksta sa greškom i
  - eventualno pogrešni deo teksta
  - radi lakšeg snalaženja u linijama programskog teksta, uputno je kao izlaz sintaksne analize predvideti i programski tekst sa rednim brojem svake linije na njenom početku

mC Parser

- primer - parsera za mC gramatiku koji rukuje sintaksnim greškama