Parser – primer kalkulatora

- Primer gramatike za jednostavni kalkulator

```
lines →  ε
         |lines NEWLINE
         |lines e NEWLINE
e     →  e "+" NUMBER
         |e "–" NUMBER
         |NUMBER
```

(podrazumeva se da su **NUMBER** i **NEWLINE** simboli)

- Ulaz:

```
5 + 2
1 – 3
2 * 5
```

Izlaz:

```
 7
-2
syntax error
```

Parser – primer kalkulatora

- Kalkulator definisan prethodnom gramatikom može biti u obliku parsera čije ponašanje opisuje odgovarajuća Bison specifikacija

- primer: **`calc1`**

- U Bison specifikaciji je navedeno da kalkulator prihvata jedan izraz u jednoj liniji (prvo pravilo)
- Druga grupa pravila opisuje izraze

Parser – primer kalkulatora

- U Bison specifikaciji je navedeno da kalkulator prihvata jedan izraz u jednoj liniji (prvo pravilo)
  - i kada prepozna izraz odštampa njegovu vrednost (**`printf`** štampa vrednost drugog pojma e)
- U drugoj grupi pravila je prikazano računanje izraza
- Podrazumeva se da vrednost simbola **`NUMBER`** vraća skener u globalnoj promenljivoj **`yylval`**

- primer: **`calc2`**

Parser – Bison – oporavak od greške

- Kada otkrije grešku, podrazumeva se da LR parser pozove funkciju **`yyerror`** radi ispisivanja poruke "*syntax error*" i da zatim završi parsiranje
- Međutim Bison dozvoljava da se umesto podrazumevajućeg obavi korisničko tretiranje greške
- Bison podržava oporavak od grešaka nastalih za vreme pokušaja redukcije po nekom pravilu tako što dozvoljava da se u takvo pravilo uvede alternativa koja omogućuje redukciju nakon pojave greške
  - takva alternativa sadrži Bison ključnu reč **`error`** i niz pojmova i/ili simbola koji treba da budu otkriveni nakon pojave greške da bi u slučaju greške redukcija bila moguća
    - neposredno iza **`error`** obavezno sledi simbol
  - na ovaj način se može ignorisati niz ulaznih simbola koji sadrže grešku i nastaviti sintaksna analiza iza takvog niza simbola

Parser – Bison – oporavak od greške

- Na primer, da bi se ignorisali svi simboli iz pogrešne linije kod jednostavnog kalkulatora potrebno je u prvo pravilo dodati alternativu

```
|   lines error NEWLINE
        { yyerror("reenter last line:"); yyerrok; }
```

- Kada parser naiđe na grešku u liniji sa izrazom on primeni prethodno pravilo
  - pre toga izmeni stanje na steku tako da može da prepozna alternativu pravila koja omogućuje oporavak od greške
  - podrazumeva se da je **`yyerror`** funkcija koja prikazuje poruku u slučaju greške, a
  - **`yyerrok`** je makro koji signalizira LR parseru da je greška obrađena i da može da nastavi sa parsiranjem
- primer: **`calc3`**

Parser - dvosmislene gramatike

- Gramatika je dvosmislena (*ambiguous*) ako razna izvođenja dovode do istog niza simbola
- Kod dvosmislenih gramatika za isti niz simbola postoje bar dva različita stabla parsiranja
- Dvosmislene gramatike su problematične, jer dozvoljavaju različita tumačenja jednog-istog niza simbola

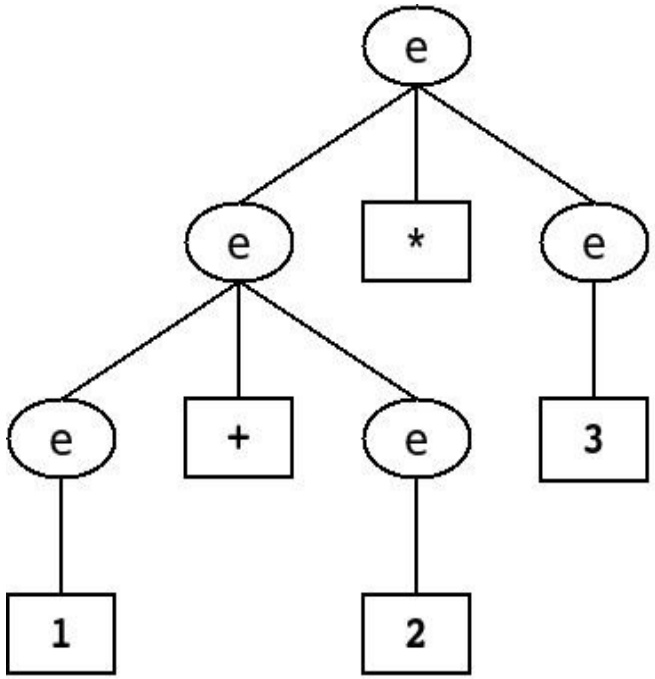
Parser - dvosmislene gramatike

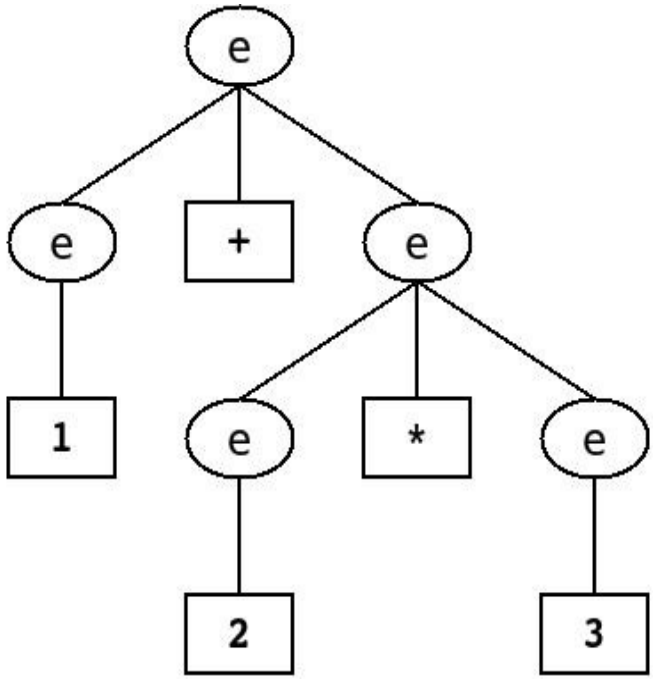
- Primer dvosmislene gramatike
- Gramatika za kalkulator koji podržava još i množenje, deljenje i unarni minus

```
e  →  e + e
   |  e - e
   |  e * e
   |  e / e
   |  -e
   |  NUMBER
```

Parser - dvosmislene gramatike

- Primer dvosmislenosti: dva različita stabla parsiranja za isti niz simbola: `1 + 2 * 3`
- Izbor levog ili desnog drveta zavisi od prioriteta operatora





Parser - dvosmislene gramatike

- Dvosmislenosti redosleda primene operatora, zahtevaju da se parseru saopšti redosled njihove primene *(associativity* i *precedence)*
- Redosled primene operatora s leva u desno se zadaje u obliku

```
%left    '+' '-'
```

(primer se odnosi na operatore sabiranja i oduzimanja), a s desna u levo u obliku

```
%right   UMINUS
```

(primer se odnosi na unarni minus)

- Prvo se navode manje prioritetni operatori

```
%left    '+' '-'
%left    '*' '/'
%right    UMINUS
```

(u prethodnom primeru najprioritetniji operator je unarni minus)

- Prioritet operatora se pridružuje pravilu ako se iza pravila navede oznaka **%prec** i zatim dotični operator
- primer kompletnog kalkulatora: **calc4**

Gramatika mC programskog jezika

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

Gramatika mC - simboli

letter

→

“a”|“A”|“b”|“B”|“c”|“C”|“d”|“D”|“e”|“E”|“f”|“F”|“g”|“G”|“h”|“H”
|“i”|“I”|“j”|“J”|“k”|“K”|“l”|“L”|“m”|“M”|“n”|“N”|“o”|“O”|“p”|“P”
|“q”|“Q”|“r”|“R”|“s”|“S”|“t”|“T”|“u”|“U”|“v”|“V”|“w”|“W”|“x”|“X”
|“y”|“Y”|“z”|“Z”

digit

→ “0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”

identifier

→ letter (letter | digit)*

int_constant

→ digit +

Gramatika mC - pojmovi

program

→ *function_list*

function_list

→ *function*

→ *function_list function*

function

→ *type* **identifier** “**(**” “**)**” *body*

type

→ “**int**”

Gramatika mC - pojmovi

body
→ “**{**” *variable_list statement_list* “**}**”
→ “**{**” *statement_list* “**}**”

variable_list
→ *variable* “**;**”
→ *variable_list variable* “**;**”

variable
→ *type* **identifier**

statement_list
→ ε
→ *statement_list statement*

Gramatika mC - pojmovi

statement
→ *compound_statement*
→ *assignment_statement*
→ *if_statement*
→ *return_statement*

compound_statement
→ “**{**” *statement_list* “**}**”

assignment_statement
→ *identifier* “**=**” *num_exp* “**;**”

Gramatika mC - pojmovi

num_exp
→ *exp*
→ *num_exp* “**+**” *exp*
→ *num_exp* “**-**” *exp*

exp
→ **constant**
→ **identifier**
→ *function_call*
→ “**(**” *num_exp* “**)**”

constant
→ **int_constant**

Gramatika mC - pojmovi

function_call
→ **identifier** “**(**” “**)**”

if_statement
→ *if_part*
→ *if_part* “**else**” *statement*

if_part
→ “**if**” “**(**” *rel_exp* “**)**” *statement*

rel_exp
→ *num_exp* “**==**” *num_exp*
→ *num_exp* “**!=**” *num_exp*

Gramatika mC - pojmovi

return_statement

→ “**return**” *num_exp* “**;**”

- Komentari:
  - višelinijski: `/* a comment */`
  - jednolinijski: `// a comment`
- Razmak i kraj linije imaju funkciju separatora simbola

mC – primer programa

```
int main() {
   int a;
   int b;
   a = 55;

   if(a == 0)
      b = 1;
   else
      b = 0;

   return b;
}
```

```
int f2() {
   int x;
   x = 2;

   return x * x;
}

int main() {
   int a;
   a = f2();
   return a;
}
```