yacc (yet another compiler-compiler)

yacc on tööriist I kirjeldamiseks mingile P-le (kompilaatorile)
yacc'i on kasutatud C, FORTRAN'i, Pascal'i, kalkulaatorite keelte,
jne. kompilaatorite koostamiseks
yacc - süntaksi analüüsi P-de (SA) koostamiseks
sisend: süntaksi spetsifikatsioon
väljund: süntaksi analüüsi P C-s või RATFOR-is
yacc - I-struktuuri viimiseks P-i kujule (C-P)
yacc'i programmeerija määrab I struktuuri ja teeb I-protsessi
yacc-spetsifikatsiooni (YS), mis sisaldab:
• I struktuuri reeglid
• koodi, mis kuulub täitmisele reeglite äratundmisel
• madalama astme P, mis teostab I-voo sisestamist (***yylex***)
SA poole pöörduv P analüüsib lekseemite sisendvoogu, mis tuleb
yylex-st; lex ja yacc on seotud; (yylex() võid ka ise kirjutada)
vt. pilti
yacc organiseerib lekseemid vastavalt grammatika reeglitele
kui yacc tuvastab I-jada, mis allub mingile reeglile, siis täidetakse
sellele reeglile vastav kood (tegevus)
tegevused tagastavad väärtusi ja opereerivad teiste tegevuste väärtustega
yacc genereerib tegevusi ja väljastusi C-s
paljud süntaksi konventsioonid vastavad C-le
YS põhiosa on gramm. reeglite kogum
iga reegel kirjeldab lubatud struktuuri ja annab talle nime
näiteks grammatika reegel:
kuupaev: paev '.' kuu aasta '.' 'a' '.' ;
siin *kuupaev*, *paev*, *kuu* ja *aasta* on sisendprotsessi struktuurid
paev, *kuu* ja *aasta* on defineeritud mujal
'.', 'a' - terminaalsed sümbolid
seega rahuldab reeglit järgmine I :

I ei pruugi vastata YS-le; I vead tuleks avastada nii ruttu, kui see teoreetiliselt on võimalik; ↓võimalust arvutada vigaste andmetega, võimaldab neid uuesti sisestada või jätkata sisestust, jättes vigased andmed vahele

mõnikord ei saa yacc SA koostamisega hakkama (YS on vasturääkiv,…), LA tuleb teha võimsamaks või mõned reeglid ümber teha

YS koostamise reeglid

• YS struktuur sama, mis LS-l (3 osa, …); ‘ ‘, ‘\t‘, ‘\n‘ ignoreeritakse, kommentaarid - /*… */

• reegli üldkuju:

```
A       :  keha  ;
```

A - nimi

keha - 0 või pikem nimede ja literaalide jada

: ; - kirjavahemärgid

: - reegli algus

; - reegli lõpp

• YS-S 2 liiki nimed:

1) lekseemide nimed
2) reeglite nimed (nn. mitteterminaalsed)

nime pikkus - suvaline, võib sisaldada tähti, numbreid, ‘.‘-e, ‘_‘-e

• literaal - sümbol ´´-de vahel, ‘\‘ - escape-sümbol, samuti kõik C omad

• kui mitu reeglit sama vasaku poolega (sama nimega), siis ‘|‘ võimaldab mitte kirjutada vasakut poolt, ‘;‘ võib enne ‘|‘ ära jätta

```
A       :  B  C  D  ;
A       :  E  F  ;
A       :  G  ;
```

asemel võib

```
A       :  B  C  D
        |  E  F
        |  G
        ;
```

• kui nimele vastab tühi string, siis analoogiliselt LS-ga:

```
A       ;
```

• lekseemide nimed peavad olema deklareeritud; lihtsaim on seda teha dekl. osas:

```
%token NIMI1 NIMI2 …
```

reegli nimi peab esinema vähemalt 1 reegli vasakul pool
on soovitav, et LA oleks YS F-i osa

• eritähendus on dekl. ***start***, mis määrab kõige 1. reegli

```
%start  symbol
```

teda kasut. SA töö lõpetamiseks (vt. näidet)

tegevused

iga reegliga saab siduda tegevusi; tegevused võivad tagastada väärtusi ja kasutada eelmiste tegevuste väärtusi
tegevus on suvaline C lause(d), mis paikneb(-vad) {}-de vahel

```
A       : ´(´  B  ´)´
        {   hello (1,  “abc“ ) ;  }            ja
XXX     :    YYY   ZZZ
        {          printf(“teade\n“) ;
                   flag  = 25;         }
```

on reeglid koos tegevusega
LA ja tegevuste vahelise sideme hõlbustamiseks on tegevuste lauseid natuke muudetud - ‘$‘ kasut. signaalina pseudomuutujatest
väärtuse tagastamiseks omistab tegevus pseudomuutujale ‘$$‘ väärtuse

```
{       $$ = 1;  }          tegevus, mis tagastab väärtuse 1
```

eelmiste tegevuste ja LA väärtuste kasutamiseks saab tegevus kasutada pseudomuutujaid $1, $2,… mis viitavad reegli parema poole komponentide väärtustele lugedes vasakult paremale

A : B C D ; $2 - C tagastatud väärtus
avaldis : ´(´ avaldis ´)´ ;
tagastatakse sulgudes oleva avaldise väärtus, mida võib näidata:
avaldis : ´(´ avaldis ´)´ { $$ = $2 }
vaikimisi on reegli väärtuseks $1 väärtus (1. elemendi väärtus)
A : B ; automaatselt pannakse $$ = $1
(tegevusi võib teha ka enne reegli lõppu)
paljudes rakendustes tehakse väljund mitte ainult otseste tegevuste abil (näit. tehakse grammatika analüüsi puu)
olgu näiteks C-funktsioon ***node***(L, n1, n2), mis loob tipu märgendiga L ja alamtippudega n1 ja n2 ja tagastab loodud tipu indeksi
siis saab ehitada grammatika analüüsi puud järgmist tüüpi tegevusega
avaldis : avaldis ´+´ avaldis
{ $$ = node(´+´ , $1, $3) ; }
programmeerija saab kasutada tegevustes oma def. muutujaid
selleks tuleb need dekl. osas deklareerida (defineerida); need muutujad on globaalsed, s.t. nad on kasutatavad nii tegevustes kui LA-s
%{ int muutuja = 0; %}
‘muutuja ‘on kasutatav kõikides tegevustes
LA ise kasutab ainult nimesid, mis algavad ‘yy‘-ga

YS ja LS

YS erineb LS-st peamiselt reeglite formaadi poolest
LS: reeglid on regulaarsed avaldised
YS: reeglite ahelad, mis võivad viidata iseendale (rekursioon)
näiteks kask võib olla defineeritud järgmiselt:

```
kask:   SET BIT numb ONCMD ENDCMD
        |
        SET BIT numb OFFCMD ENDCMD
        ;
```

yylex-i poolt väljastatavad lekseemid tähistatakse harilikult suurtähtedega, YS-s kasutatavad identifikaatorid aga väiketähtedega
eeltoodud reeglis saab sarnased alternatiivid kokku võtta

```
kask:   SET BIT numb onoff ENDCMD;
```

tuues sisse järgmised reeglid:

```
onoff:  ONCMD
        |
        OFFCMD
        ;
numb:   NUMBER;
```

YS koostatakse printsiibil “ülalt alla“
ülalpool asuvad üldisemad reeglid, mis hargnevad ja arenevad allapoole
vahetult lekseeme puudutavad reeglid asuvad lõpus
kuna yacc-is ühed reeglid võivad olla avaldatud teiste reeglite abil, on kasutusel spetsiaalne mehhanism muutujate väärtuste vahetamiseks reeglite vahel (kask, onoff)
alumise nivoo reegel (numb) võib anda vastava väärtuse üles (kask) omistades selle muutujale $$:

```
numb:   NUMBER
        {$$ = yylval; }
        ;
```

SA töö

SA kujutab endast lõplikku automaati, millel on nn. pinu (stack)
ta on võimeline sisestama ja meeles pidama järgmist lekseemi
tal on 4 tegevust: nihe (shift), (taandus) reduce, vastuvõtmine (accept), viga (error)
LA põhiosa - yylex() täisarvuline funkts.
tagastab täisarvu - lekseemi #, mis näitab lekseemi tüüpi
kui lekseemiga on seotud väärtus, siis see tuleb omistada extern-muutujale *yylval* (int)
LA ja SA peavad teineteisest aru saama

lekseemide #-d võib valida kas yacc või programmeerija; mõlemil juhul kasutatakse C ‘# define‘ mehhanismi; see võimaldab #-d tagastada nimedena
NB! ‘bug‘ - nimi, mida kasutab C või LA (if, while,error,…)
reeglid on kahemõttelised, kui I-jada saab tõlgendada mitmel viisil
näiteks:
avaldis : avaldis ´-´ avaldis
ei määra üheselt, kuidas kõiki keerulisi I-d tuleb struktureerida
kui sisendis on
avaldis - avaldis - avaldis
siis see reegel lubab I struktureerida nii, et 1.-na taandatakse
(avaldis - avaldis) - avaldis
või nii, et 1.-na nihutatakse
avaldis - (avaldis - avaldis)
vastavalt vasak(parem-)poolne assotsiatiivsus;
antud juhul on tgemist nihe/taandus konfliktiga; on võimalik, et on 2 legaalset taandust - taandus/taandus konflikt
yacc avastab need ära, kuid teeb SA järgmiste reeglite põhjal:
1) nihe/taandus konflikti korral tehakse enne nihe
2) taandus/taandus konflikti korral taandatakse varasema (eespool) oleva reegli järgi
konfliktid võivad tekkida I-vigade või loogikavigade tõttu
yacc väljastab 2 tüüpi konfliktide arvu

```
lause   :  IF   ´(´  loogav  ´)´  lause
        |  IF   ´(´  loogav  ´)´  lause  ELSE   lause
        ;
```

see on kahemõtteline, kuna I
IF (l1) IF (l2) lause1 ELSE lause2
on struktureeritav 2 moodi:

```
IF  (   l1  )        {
    IF   (   l2  )    lause1   }
ELSE   lause2
```

või

```
IF  (   l1  )        {
          IF   (   l2  )    lause1
          ELSE   lause2
}
```

konfliktide teated saab, kasutades võtit -v (verbose)
antud juhul:
23: shift/reduce conflict (shift 45, reduce 18) on ELSE
… (järgneb ***y.output*** P-lõik)

prioriteedid

probleem: teatud juhtudel - (aritmeetiliste avaldiste gramm. analüüs)
tradits. konfliktide lahendamise reeglid ei aita
olgu:
avaldis : avaldis   OP   avaldis
avaldis : UN   avaldis
kõikide binaarsete ja unaarsete operaatorite jaoks
tulemus - kahemõtteline grammatika paljude konfliktidega
kontra - kõikide operaatorite prioriteedid, vasak- ja parempoolne assotsiatiivsus
dekl. osas omistatakse lekseemidele prioriteedid ja assots.
kasutades yacc'i võtmesõnu %prec, %left, %right, %nonassoc
%prec peab olema vahetult pärast gramm. reeglit
samal real on kõikidel lekseemidel sama tase, read on prioriteedi suurenemise järjekorras

```
%left    ´+´        ´-´
%left    ´*´        ´/´
```

kirjeldavad 4 aritm. tehte prioriteeti ja assotsiatiivsust

%nonassoc - kasut. näiteks FORTRAN'i operaatori .LT. korral (A .LT. B .LT. C) - ei ole lubatud FORTRAN'is

näiteks:

```
%right  ´=´
%left   ´+´     ´-´
%left   ´*´     ´/´
%%
avaldis :       avaldis ´=´ avaldis
        | avaldis ´+´ avaldis
        | avaldis ´-´ avaldis
        | avaldis ´*´ avaldis
        | avaldis ´/´ avaldis
        | ´-´ avaldis       %prec   ´*´
        | NIMI
        ;
```

olgu I-s:

```
a = b = c*d - e - f*g
```

siis see struktureeritakse:

```
a = ( b = ( ( ( c*d ) - e ) - ( f*g ) ) )
```

veafunktsioon

```
$ cat yyerror.c
#include <stdio.h>
yyerror(s) char *s;
   {
   fprintf(stderr,"%s\n", s);
   }
```

näide

olgu struktuur, mida tuleb initsialiseerida vastavate väärtustega
protsessi interaktiivne juhtimine toimub vastava käsukeele abil, milles on lubatud näiteks järgmised käsud:
set bit 3 on
set 197
set bit 4 off ; set bit 0 on
vastav YS on põhiline selle rakenduse jaoks;
ta opereerib mingi muutujaga; ta omistab 0 või 1 tema bittidele
analoogiliselt lex-S-ga sisaldab YS 1.-s osas kirjeldused, 2.-s aga reeglid; eraldaja “%%“; viimane, 3.-s osa kujutab endast C main-f-i, mis pöördub süntaksi analüüsiks genereeritud f-i yyparse() poole
kirjelduste osa on C-s (“%{“ ja “%} “ vahel)

```
%{
/*
 * yacc - spetsifikatsioonide F
 *      1. osa - kirjeldused
 */
int testvar = 0;
int yylval;
#define Off  0
#define On  1
%}
%token SET,BIT,ONCMD,OFFCMD
%token NUMBER, ENDCMD, UNKNOWN
%%
/*
 *      2. osa - reeglid
 */
session:        /*  1. reegel  */
                |
                kasud
                ;
```

```
kasud:      kask    /* 2. reegel  */
            |
            kasud kask
            ;
kask:          ENDCMD  /* 3. reegel    1. alternatiiv  */
                  {  /* tühi käsk */ }
            |
            error   ENDCMD  /* 3. reegel    2. alternatiiv  */
            |
            SET BIT numb onoff ENDCMD  /* 3. alternatiiv */
                  {   if (($3 <= 15) && ($3 >= 0))                    {
                        {   if ($4 == Off)
                              testvar = testvar & ~(1 << $3);
                            else
                              testvar = testvar | (1 << $3);
                         }
                       else printf("Vale biti number: %d\n",$3);
                       printf("Testvar - %o\n",testvar);
                  }
            |
            SET numb ENDCMD /* 3. reegel - 4. alternatiiv */
                  {    testvar = $2;
                       printf("Testvar - %o\n",testvar);
                  }
            ;
numb:   NUMBER  /* 4. reegel */
           {  $$ = yylval;       }
      ;
onoff:  ONCMD   /* 5. reegel - 1. alternatiiv */
            { $$ = On; }
      |
      OFFCMD  /* 5. reegel - 2. alternatiiv */
            { $$ = Off; }
      ;
```

```
%%
/*
 * 3. osa
 * abifunktsioonid
 */
main()
{
yyparse();
}
```

salvestame YS F-i ‘yaccdemo.y‘ ja kirjutame f-i yyerror(), mille salvestame F-i ‘yyerror.c‘ (vt. ette)

$ yacc -d yaccdemo.y

teeb F-i ‘y.tab.c ‘

yacc võib teha veel 2 F-i: ‘y.tab.h‘ - kui seda ei tehtud yylex’i juures ‘y.output‘ - LA olekute ja konfliktide F (võti (-v) - verbose)

transleerime:

$ cc -o yaccdemo y.tab.c lex.yy.c yyerror.c

katsetus:

$ yaccdemo

santra> yaccdemo

```
set bit 3 on
Testvar - 10
bit set 4 on
syntax error
set 0
Testvar - 0
set bit 15 on
Testvar - 100000
set bit 15 off
Testvar - 0
set 65
Testvar - 101
set bit 16 on
Vale biti number: 16
Testvar - 101
set 1997
Testvar - 3715
set 0
Testvar - 0
```