Unix’i loeng 03.04.97 1.

yacc ja lex

yacc kompileerivate programmide generaator
lex leksika analüsaatorite generaator
mõnikord vaja spetsiaalseid käsukeeli (näiteks C, shell, make’i seoste kirjeldamise keel)
mõlemad generaatorid koostavad P-d 2 F-i abil:
• shabloonP-i F
• spetsifikatsiooniF
1. sisaldab P-i karkassi C-s
2. koostab programmeerija
tulemusena saadakse vastavalt etteantud nimedega C-P-d:
lex.yy.c (lex) y.tab.c (yacc)
seega: lex ja yacc on tegelikult tekstiprotsessorid, mis shabloonF-i ja spetsifikatsiooniF-i abil koostavad C-P-d
spetsifikatsiooniF-d vormistatakse erikujuga reeglite hulgana
lex’i reegel annab vastavuse lekseemi (eraldatakse I-voost) ja vastava tegevuse, mis tuleb täita, vahel
tegevuseks on harilikult lekseemi tagastamine mingil kujul P-i, mis kutsus välja leksika analüsaatori (LA)
lekseem, mida tuleb tuvastada vastava tegevuse täitmiseks, antakse spetsifikatsioonis samuti shabloonina (regulaarse avaldisena)
yacc’i sisendinfoks on keele grammatika, mis esitatakse erikujuga reeglite abil (reeglid on sarnased laiendatud BN kujule)
iga reegliga võivad olla seotud tegevused (pöördumine C-P-i poole,..)

leksika ja süntaksi analüüs

kompilaatorite koostamine - iseseisev õppeaine (keeruline)
vaatleme ainult spetsifikatsiooniF-de koostamist
need F-d on järgmise struktuuriga:

definitsioonid **03.04.97** **2.**
%%
reeglid
%%
funktsioonid

keerulise käsukeele analüüs 2:
• leksika analüüs
• süntaksi analüüs
1.-s tuvastatakse madalat järku objektid: arvud, tehtemärgid, spets. sõnad
2.-s faasis keerulised (operaatorid, laused, …)
piir mitte väga range (kogemuslikult)
P-d, mis seda realiseerivad, leksika ja süntaksi analüsaatorid (LA, SA)
sisendinfo: sümbolite voog (terminalilt, failist, …)
LA vaatab läbi sisendinfot vastavalt etteantud reeglitele ja tuvastab objekte; leides objekti, loob ta tema tüübi kirjelduse, mida nim. lekseemiks; näit. kui LA sisendile saabub:
25*(16/2) + 15,
luuakse lekseemite jada
NOLNONRON
kus: N - arv
O - operatsioon
L - vasak sulg
R - parem sulg
sellest saab aru SA
leksika analüüsi faas on sisendinfo filtriks juhuslike vigade vastu (probell vales kohas,…)
SA vastutab kõrgemal tasemel
kui saadi NOLNONRON, siis SA peab kontrollima, kas see on korrektne ning tegema vastavad tegevused

lex **03.04.97** **3.**

lex on LA-te genereerimiseks; genereeritakse F nimega ***lex.yy.c***
ta sisestab LA spetsifikatsiooni ja koostab P-i C-s või nn. RATFOR-is (lex-alamP); põhiline selles F-s on funktsioon ***yylex()***, mis analüüsib I-voogu ja eraldab sealt lekseemid
LA töölepanemiseks peab ta olema kompileeritud ja ühendatud teiste P-dega, vt joonis ‘leks.sg‘
NB! lex (ja yacc) teevad ainult alamP-d, mitte aga lõpetatud analüsaatorid; see lubab P-i lisada teksti, mis vastab kindlale rakendusele
leksika reeglid esitatakse lex-spetsifikatsioonide osas nn. regulaarsete avaldistena (RA)
lex-spetsifikatsioonide osa on RA ja nendele vastavate P-i fragmentide tabel, mis lex’i poolt teisendatakse P-ks

- RA on lekseemide (stringide) tuvastamiseks I-voos
- P-i fragmendid täidetakse, kui on tuvastatud lekseem

lex-spetsifikatsioon (LS)

spetsifikatsioonide üldkuju:

```
definitsioonid
%%
reeglid
%%
funktsioonid
```

esimene %% on kohustuslik, 2. ei ole, kuna definitsioonid ja funktsioonid võivad puududa
absoluutselt minimaalne LS:
%% (ei ole definitsioone ega reegleid)
kopeerib I-i O-sse muutmata kujul

03.04.97 4.

lex-i RA-d

RA-d on sarnased U-i redaktorite (ed, ex, vi) avaldistele, mõnedel sümbolitel on seal eritähendus:

. mistahes sümbol (excl. reavahetus - \n)
\n reavahetuse sümbol
\t tabulatsiooni sümbol
operaatorid:
\ ^ ? * + | $ / %
[] {} () <> “ - .
kui vaja muuta sümboli eritähendust, siis \-ga või “ “-dega
tähtsamad reeglid:
[] määrab sümbolite klassi, mis on seal sees
[abc] a, b või c
“-“ kasut. mistahes sümboli näitamiseks leksiliselt järjestatud jadast
[a-z] mistahes täht a..z
[+-0-9] kõik numbrid ja märgid + ja -
[]-s ainult 3 sümbolil eritähendus \, -, ^
0-9 määrab ära regulaarse avaldise, mis koosneb ühest numbrist
* näitab, et sümbol või sümbolite klass esineb 0 või rohkem *
t* t mistahes (incl. 0) arv * (, t, tt, ttt, tttt, …)
abc* mistahes arv korda abc
[a-z]* mistahes arv * mistahes ladina tähte
+ näitab, et sümbol või sümbolite klass esineb 1 või rohkem *
f+ 1 või rohkem f-i (f,ff,fff,ffff,…)
[0-9]+ vastab 0-st pikemale numbrite jadale
sümbolite valikuks kasut. operaatoreid ? / | ^
/ ab/cd - ‘ab‘ arvestatakse ainult siis, kui talle järgneb ‘cd‘
| valik alternatiivide vahel ab|cd - ‘ab‘ või ‘cd‘
? mittekohustuslik sümbol ?t - mittekohustuslik t
?[a-z]* - mistahes arvu ladina tähtede ees võib olla ‘‘

$ võtta rea viimane sümbol x$ - võtta x, kui ta on rea viimane
abc$ - võtta ‘abc‘, kui ta lõpetab rea
^ võtta rea 1. sümbol x^ - võtta x, kui ta on rea 1.
abc^ - võtta ‘abc‘, kui ta alustab rida
[a-z]^ - võtta rea 1. sümbol, kui ta on ladina väiketäht
kui ‘^‘ on avaldise ees või [] sees, siis tähendab ta täiendit
[^abc] - mistahes sümbol, v.a. a,b,c [] sees peab ta olema 1.
x{n,m} x esineb n..m * n,m - naturaalarvud, m>n
x{2,7} - x esineb 2..7 *
{nimi} ‘nimi‘ asendatakse tema delaratsiooniga lex-P deklaratsioonide osast

```
TAHT        [A-Za-z]
NUMBER      [0-9]
IDENT       {TAHT}({TAHT}|{NUMBER})*
%
{IDENT}  printf(“\n%s“,yytext);
```

funktsioonis ***lex.yy.c*** on deklareeritud: extern char yytext[];

lex-i tegevused

igale reeglile saab vastavusse panna mingi tegevuse, mis on kirjutatud väljundkeeles:

```
[ \t]+$   ;        kustutab kõik tühikud ja tab’id rea lõpus
0-9+      {printf(“leiti arv!\n”);}
```

kui ***lex*** tuvastab lekseemi, siis täidab ta etteantud tegevuse
vaikimisi (;) kopeerib I-i O-sse

```
[ \t\n]    ;        ignoreerib probelli, tab’i ja reavahetust
```

keerulisematel juhtudel on tihti vaja teada tegelikku teksti, mis tuvastati
näiteks [a-z]+ ***lex*** jätab selle välisesse (extern) sümbolite massiivi *yytext* seega

```
[a-z]+   printf(“%s“, yytext)      väljastab stringi yytext’s
```

üldjuhul, kui LA on translaatori osa, peab ta objekti avastamisel tagastama kindla väärtuse
kui on määratud identifikaator “NUMBER”, siis järgmine rida lex-spetsifikatsioonis tekitab koodi, mis näitab sisendvoos arvu:

```
0-9+    {return(NUMBER);}
```

aga arvu väärtus? ilmselt peab tekitatud kood andma ka sellele vastuse
lex võimaldab teksti kui objekti omistamist sümbolmuutujale “yytext“, mille C standardfunktsioon ***atoi*** teisendab täisarvuks; seega:

```
0-9+    {
            yylval=atoi(yytext);
            return(NUMBER);
        }
```

see spetsifikatsioon võimaldab juurdepääsu nii objekti väärtusele kui objekti tüübile

```
yylval   int muutuja
yytext   char muutuja
```

see tekst vastab paljudele reaalsetele lex-spetsifikatsioonidele
mõnikord vaja teada lõpus, mis tuvastatud
lex võimaldab lugeda sümbolite arvu, mis tuvastati - yyleng

```
yytext[yyleng-1]        viimane sümbol tuvastatud stringis
```

kui tahad teada, mitu sõna ja mitu sümbolit:

```
[a-zA-Z]+       {words++; chars += yyleng}
```

mõnikord vaja funktsioone:

```
yymore();       lisab yytext’i lõppu järgmise tuvastatud stringi
yyless(n);      hetkel vaja ainult n sümbolit; ülejäänud tagastatakse
                I-voogu
yywrap()        tagastab ‘1‘ I-voo lõpus, muidu 0
reject()        viib sümboli tagasi I-voogu ja kutsub välja korduva
                analüüsi (‘go do the next alternative‘)
```

03.04.97 7.

lex lubab juurdepääsu ka I/O protseduuridele:

input()	tagastab järmise sisestatava sümboli
output(c)	kirjutab sümboli c O-sse
unput(c)	viib c tagasi I-voogu

reeglid kujutavad tabelit,kus vasakus veerus on RA-d,
paremas - tegevused (P-i osad)
integer printf(“leiti sõna INT“);
avaldise lõpu määrab ära probell või tab
kui tegevus on 1 C lause, siis on ta lihtsalt paremas veerus,
kui on mitu lauset või lause läheb järgmisele reale, siis tuleb ta panna sulgudesse
prioriteedid:
1) eelistatakse pikemat kokkulangevust
2) võrdse pikkuse korral eelistatakse eespool olevat reeglit
integer t1;
[a-z]+ t2;
kui sisestada integers, siis t2
kui sisestada integer, siis t1, kuna eespool
kui sisestada int, siis t2
.* on ohtlik, kuna näiteks ‘‘-de vahel oleva stringi tuvastamisel reegel
‘.*‘ tundub olevat hea, kuid sisestades
‘ab‘ string ‘cd‘ string
leiab
‘ab‘ string ‘cd‘
mida arvatavasti ei tahetud
‘[^‘\n]*‘

definitsioonid

muutujate kirjeldamine: kas def.-de või reeglite osas
reeglid teisendatakse P-ks
mistahes spetsifikatsiooni osa, mida ei tuvastata, kopeeritakse P-i
•def.-d ***lex***-le tuuakse enne 1.-st %%
iga rida, mis ei ole %-de vahel ja mis algab 1.-s veerus, on ***lex***-i def.

nimi	väärtus
D	[0-9]
%%	
{D}+	printf(“taisarv“)

näide:

olgu tarvis tuvastada järgmised objektid:
♦ arvud (numbrite jadad)
♦ sõnad “set”, “bit”, “on”, “off”
♦ uue rea algus või semikoolon
ignoreerida probelle ja tabulatsiooni märke
määrame tagastatavad lekseemid; kasutame nimesid:

SET	BIT	ONCMD	OFFCMD	NUMBER

ENDCMD tagastatakse siis, kui leitakse uue rea algus või ;
UNKNOWN tagastatakse siis, kui leitakse midagi, mis ei vasta reeglitele
toodud lekseemite listi paigutame faili ”y.tab.h”

```
# define SET 257
# define BIT 258
# define ONCMD 259
# define OFFCMD 260
# define NUMBER 261
# define ENDCMD 262
# define UNKNOWN 263
```

lekseemite konkreetsed väärtused on valitud nii, et nad ei lõiku ASCII sümbolite hulgaga

kui seal on C konstruktsioonid, siis peavad nad olema “%{” ja “%}” vahel (vt. näidet)

```
%{
/*
 * lex - spetsifikatsioon arvude, 4 sona
 * ja eraldajate tuvastamiseks
 */
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+  {                       /* 1. reegel */
        yylval = atoi(yytext);
        return(NUMBER);
        }
;       return(ENDCMD);         /* 2. reegel  */
\n      return(ENDCMD);         /* 3. reegel  */
set     return(SET);            /* 4. reegel  */
bit     return(BIT);            /* 5. reegel  */
on      return(ONCMD);          /* 6. reegel  */
off     return(OFFCMD);         /* 7. reegel  */
[ \t]+  ;                       /* 8. reegel  */
.       return(UNKNOWN);        /* 9. reegel */
```

8. reegel - tabulatsiooni sümbolite ja probellide läbilaskmine

9. reegel - kasut. siis, kui ühtki eelmist objekti ei leitud

olgu toodud lex-spetsifikatsioon on failis ‘lexdemo.l’, siis käsuga

```
$ lex lexdemo.l
```

luuakse fail ‘lex.yy.c’, mis sisaldab funktsiooni “yylex()” C-s

nimi ‘lex.yy.c’ on standartne lex’i jaoks (analoogia ‘a.out’)

ObjektF-i saamiseks:
$ cc -c lex.yy.c
sisaldab yylex(), mõeldud töötamiseks koos ***yacc***-iga
yylex() testimiseks koostame C-s kirjutatud P-i:

```
#include "y.tab.h"
int yylval;
extern char yytext[];
/*
 *      demo
 * kutsub välja yylex() märkide proovimiseks
 */
main()
{
int token;
while (token = yylex())
        switch (token)
        {
        case NUMBER:     printf("Number: %d\n",yylval); break;
        case SET:        printf("Set\n"); break;
        case BIT:        printf("Bit\n"); break;
        case ONCMD:      printf("On\n"); break;
        case OFFCMD:     printf("Off\n"); break;
        case UNKNOWN:    printf("Unknown: %s\n",yytext); break;
        case ENDCMD:     printf("End marker\n"); break;
        default:         printf("Tundmatu märk: %d\n",token); break;
        }
}
```

03.04.97 11.

seega on meil P, mis testib
$ cc lextst.c lex.yy.o
katseta!
$ a.out sisestame
set bit 5 on;set20
Set
Bit
Number: 5
On
End marker
Set
Number: 20
End marker
set bit 3 On
Set
Bit
Number: 3
Unknown: O
Unknown: n