

YACC

(Yet Another Compiler - Compiler)

YACC

- Yacc program obezbedjuje generalni softverski alat za složene strukture ulaza.
- Korisnik yacc-a priprema specifikacije ulaznih struktura. Specifikacije sadrže:
 - Gramatička pravila koja opisuju ulazne strukture
 - Akcije koje se izvršavaju kada je prepoznata struktura usaglašena sa datim pravilom

Parser

- **yacc** program generiše funkciju za kontrolu ulaznih struktura.
- Ova funkcija se zove *parser* i ona poziva leksički analizator koji prepoznaje i selektuje tokene iz ulaznih struktura.
- Tokeni su organizovani u saglasnosti sa gramatičkim pravilima pomoću kojih je opisana struktura ulaza.
- Kada se prepozna neko pravilo, izvršavaju se akcije koje su u specifikaciji pridružene prepoznatom pravilu.

Yacc pravila

- Ulazna specifikacija je kolekcija gramatičkih pravila.
- Svako pravilo opisuje neku ulaznu strukturu i daje njeno ime. Na primer, izgled jednog gramatičkog pravila je:

```
datum : ime_meseca '/' dan '/' godina;
```
- `:` i `;` nemaju značaja u kontrolisanju ulaza, nego čine sintaksu samog pravila, a `'/'` se tretira kao literal u ulaznoj strukturi.
- *datum*, *ime_meseca*, *dan* i *godina* reprezentuju strukture od značaja u ulaznom procesu i mogu se definisati u samom yacc-u ili u lex-u.
- ulazna struktura: januar / 19 / 1992
usaglašena je sa datim gramatičkim pravilom.

Yacc pravila

- ime_meseca se može definisati u yacc-u pomoću sledećeg gramatičkog pravila:
ime_meseca : 'j' 'a' 'n' 'u' 'a' 'r';
ime_meseca : 'f' 'e' 'b' 'r' 'u' 'a' 'r';
...
ime_meseca : 'd' 'e' 'c' 'e' 'm' 'b' 'a' 'r';
- Leksički analizator treba samo da prepozna pojedinačna mala slova, a *ime_meseca* je neterminalni simbol koji prepoznaje yacc.
- Obično leksički analizator prepoznaje ime meseca i vraća *ime_meseca* kao token u yacc.
- Karakteri literali kao što je '/' u našem primeru, mogu se pomoću lex-a vraćati u yacc kao tokeni.

YACC specifikacija

- Specifikacija yacc-a sastoji se iz tri sekcije:
 - Sekcija deklaracija**
%%
 - Sekcija pravila**
%%
 - Sekcija potprograma**
- Sekcija pravila je obavezna, druge dve su opcione.
- Blankovi, tabulatori i newlines se ignorišu, ali se ne mogu uključiti u neko ime.

Sekcija deklaracija

- Sekcija deklaracija sadrži dve sekcije: sekcije C-deklaracija i sekcije yacc deklaracija.

```
%{  
    /* sekcija C deklaracija */  
}%  
    /* sekcija yacc deklaracija */  
%%
```

Sekcija pravila

- Sekcija pravila se sastoji iz jednog ili više gramatičkih pravila i svako pravilo je oblika:
A : TELO;
gde *A* reprezentuje neterminalni simbol, a *TELO* reprezentuje niz od nula ili više imena i literala.
- Ime se formira od slova, cifara, tački i donje crte, ali prvi znak mora biti slovo.
- Imena koja se koriste u telu gramatičkog pravila mogu reprezentovati tokene ili neterminalne simbole.
- Znakovne konstante stavljaju se u jednostruke znake navoda, a svi specijalni znaci programskog jezika C razumljivi su i yacc-u.

Sekcija pravila

- Ako u telu postoji više gramatičkih pravila, onda se za njihovo razdvajanje koristi '|'; na primer:

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

- Ekvivalentni zapis bez navodjenja '|' je:

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

- Ako neterminalnom simbolu odgovara prazan string, onda se to zapisuje pomoću gramatičkog pravila:

```
empty : ;
```

Imena

- Imena koja reprezentuju tokene moraju biti deklarirana, i to u sekciji yacc deklaracija:
`%token <ime-1> <ime-2> ...`
- Neterminali se deklariraju pomoću ključne reči `%type`.
- Startni simbol ima posebno značenje. Po default-u, startni simbol nalazi se na levoj strani prvog gramatičkog pravila u sekciji pravila.
- U sekciji deklaracija eksplicitno se deklariraju startni simbol i to korišćenjem rezervirane reči `%start`.

```
%start <ime-simbola>
```

Semantičke akcije

- Svakom gramatičkom pravilu može se pridružiti akcija koja se izvršava kada se prepozna gramatičko pravilo.
- Akcije mogu da vraćaju vrednosti i dobijaju vrednosti vraćenih pomoću prethodnih akcija.
- Akcije su instrukcije programskog jezika C.

Semantičke akcije

- Znak \$ se koristi za komunikaciju između parsera i akcija.
- Pseudo-promenljiva \$\$ reprezentuje semantički neterminal koji se nalazi na levoj strani gramatičkog pravila
- Za označavanje vrednosti vraćenih pomoću prethodnih akcija, akcija može da koristi pseudo-promenljive \$1, \$2, ... , \$n. Na ovaj način se referencira na vrednosti vraćene pomoću komponenti 1 do n koje se nalaze na desnoj strani pravila.

Primeri

- Kada parser prepozna gramatičko pravilo **prvi**, pomoću semantičke akcije `$$=1` neterminalu **rezultat** dodeljuje se vrednost 1.

```
rezultat : prvi
          { $$ = 1;};
```

- Ako je dato gramatičko pravilo:

```
rezultat : prvi drugi treci ;
```

onda `$2` sadrži vrednost neterminala drugi, a `$3` vrednost neterminala treći.

Primeri

- U pravilu:

```
expr : '(' expr ')';
```

prva komponenta je literal, tj. '(', pa je logički rezultat:

```
expr : '(' expr ')'
      { $$ = $2; }
      ;
```

Primeri

- Yacc program dozvoljava pisanje akcije i u sredini nekog pravila, a ne kao u prethodnom primeru - na kraju pravila:

```
A : B
    { $$ = 1; }
    C
    { x = $2;
      y = $3; } ;
```

x dobija vrednost 1, a y dobija vrednost koja je vraćena pomoću C.

Akcija `$$ = 1` vraća vrednost 1, a toj vrednosti mogu preko `$` mehanizma da pristupe i akcije koje se nalaze desno od navedene akcije; u našem primeru to je akcija `x = $2;` .

Promenljive

- Korisnik može da definiše i druge promenljive koje se koriste u akcijama.
- Deklaracija takvih promenljivih vrši se u sekciji C deklaracija.
- Ove deklaracije imaju globalni doseg, tako da su poznate svim akcijama u yacc-u.

Pravila prednosti i asocijativnosti kod aritmetičkih izraza

- Gramatičko pravilo:

izraz : izraz '-' izraz

je uobičajeni način specificiranja aritmetičkog izraza.

- Ako je ulaz :

izraz - izraz - izraz

onda navedeno pravilo može da struktuiru ulaz na dva načina:

(izraz - izraz) - izraz

leva asocijativnost

izraz - (izraz - izraz)

desna asocijativnost

Pravila prednosti i asocijativnosti kod aritmetičkih izraza

- Da bi se razrešio navedeni problem, uvode se pravila prednosti zajedno sa informacijama o levoj ili desnoj asocijativnosti.
- Za označavanje asocijativnosti uvode se sledeće ključne reči:

%left i %right

- Iza navedenih ključnih reči sledi lista tokena. Svi tokeni koji se nalaze u istom redu imaju istu prednost i asocijativnost. U sledećem primeru dat je opis prednosti i asocijativnosti četiri aritmetičke operacije:

%left '+' '-'

%left '*' '/'

Pravila prednosti i asocijativnosti kod aritmetičkih izraza

- Za promenu prednosti koristi se ključna reč **%prec**. Na primer,
izraz : '-' izraz %prec '*'
ovo pravilo specificira da unarni minus ima istu prednost kao i operacije množenja *.

Rekurzivna pravila

- Pravilo oblika leve rekurzivne gramatike je oblika:
ime_pravila : ime_pravila ostatak_pravila
- Veoma često se koriste sledeća pravila za pisanje specifikacija nizova i listi:
lista : clan
| lista ',' clan ;
niz : clan
| niz clan ;
- U svakom od ovih slučajeva, prvo pravilo biće reducirano samo za prvi član; drugo pravilo biće reducirano za drugi i sve ostale članove.

Tipovi podataka u yacc-u

- Po default-u vrednosti koje vraća leksički analizator i akcije u yacc-u su celobrojnog tipa.
- Yacc program podržava i druge tipove.
- Yacc deklarise stek vrednosti kao uniju različitih tipova vrednosti.
- Korisnik deklarise uniju i pridružuje imena članova unije svakom tokenu i netrinalnom simbolu koji ima vrednost.
- Kada se vrednost referencira sa \$\$ ili \$n konstrukcijom, yacc će automatski da ubaci odgovarajuće ime unije.

Tipovi podataka u yacc-u

- Deklaracija unije zadaje se u sekciji yacc deklaracija:
%union
{ <telo_unije> }
- Na ovaj način deklarise se yacc stek vrednosti i eksterna promenljiva *yyval* ima tip ekvivalentan uniji.
- Ako se yacc poziva sa -d opcijom, deklaracija unije kopira se u *y.tab.h* kao *yystype*.
- Kada je definisano *yystype*, imena članova unije moraju biti pridružena imenima terminala i neterminala.

Tipovi podataka u yacc-u

- Konstrukcija `<ime>` koristi se za označavanje imena članova unije.
- Ključna reč `%type` koristi se za deklarisanje neterminala, a tipovi neterminala određeni su imenima članova unije.
`%type <ime> izraz`
- Ime člana unije `<ime>` pridružuje se neterminalnom simbolu *izraz*.
- Ako ime člana unije sledi posle ključnih reči `%token`, `%left`, `%right` ili `%nonassoc`, onda se ime člana unije pridružuje tokenu. Na primer,
`%left <ime> '+' '-'`
vrednost koja se dobija pomoću tokena '+' ili '-' je tipa člana unije.

Tipovi podataka u yacc-u

- Ako postoji akcija unutar pravila, vrednost koja se dobija pomoću ove akcije nema a priori tip. U takvom slučaju, tip se može odrediti ubacivanjem imena člana unije između `<>` neposredno posle prvog znaka `$`.

```
Pravilo : aa
        { $<interval>$ = 3; }
        bbb
        { fun ($<interval>2); }
        ;
```

Obrada grešaka

- Obrada grešaka omogućena je preko ulazne specifikacije.
- U graničnim pravilima može se koristiti token `error`. Na primer, u pravilu oblika
start : error
parser, posle otkrivanja sintaksne greške, ostaje u stanju "ERROR"
- Ako se otkrije sintaksna greška kada je parser u stanju "ERROR", parser ne daje poruku o grešci i ulazni token se briše.
- Ako se token `error` ne specificira u gramatičkim pravilima, nakon otkrivanja sintaksne greške, prekida se proces čitanja ulaza.

Obrada grešaka

- Pravilo oblika
start : error ';' ;
znači da kada parser otkrije sintaksnu grešku, tokeni koji slede posle otkrivene greške i prethode sledećoj tački_zarezu, jednostavno se odbacuju.
- Jedan drugi oblik error pravila koji se koristi u interaktivnim aplikacijama, dozvoljava ponovo unošenje reda posle otkrivanja greške., na primer
input : error '\n'
{ (void) printf ("Ponovo unos poslednje linije");}
input
{ \$\$ = \$4;};

Obrada grešaka

- Problem u ovom pristupu je što parser, posle otkrivanja greške, mora da obradi tri ulazna tokena, pre nego što utvrdi da su oni korektni.
- Ako ponovo uneseni red sadrži neku grešku u prva dva tokena, parser briše tokene i ne daje poruku o grešci.
- Poslednji primer može se napisati i na sledeći način:

```
input : error '\n'  
      { yyerror;  
        (void) printf ("Ponovo unesi poslednji red");}  
input  
      { $$ = $4;};
```
- Izraz *yyerror* vraća parser u njegov normalni mod. Token koji se vidi neposredno posle *error* simbola je ulazni token na kojem je otkrivena greška.

Translacija i izvršavanje Yacc-a

- Za dobijanje leksičkog analizatora u C-u:
lex <ime-prog-lex>.l
- Za dobijanje sintaksnog analizatora u C-u:
yacc -d <ime-prog-yacc>.y
gde je <ime-prog-yacc>.y datoteka koja sadrži yacc specifikaciju, a yacc generiše sintakсни analizator u C-u, koji se nalazi u datoteci y.tab.c i header file y.tab.h
- Kompajliranje i linkovanje se vrši kao i za svaki drugi C program:
gcc -o <ime-prog> lex.yy.c y.tab.c -lfl

Primer

- Napisati yacc aplikaciju za celobrojnu aritmetiku. Koristiti samo operacije +, -, *, / i unarni minus.
- Osnovna notacija za pisanje gramatičkih pravila za aritmetičke izraze je sledeća:

izraz : izraz op izraz

izraz : unmin izraz

Lex specifikacija

```
%{
#include <stdio.h>
#include<string.h>
#include "y.tab.h"
extern int yyline;
%}
%%
[0-9]+ { strcpy(yylval.name,yytext);
        return(NUMBER);}

\t ;
\n yyline++;
[^0-9] return(yytext[yytextleng-1]);
%%
```

Yacc specifikacija – sekcija deklaracija

```
%{
#include <stdio.h>
#include <string.h>
int yyline;
int yylex();
}%

%start list
%union {
    int ival;
    char name[32];}

%token <name> NUMBER
%type <ival> expr stat
%left <ival> '+' '-'
%left <ival> '*' '/'
```

Yacc specifikacija – sekcija pravila

```
%%
list : stat;
list : list stat;
stat : expr '.'
      { printf("rezultat %d\n", $1); };
expr : '(' expr ')'
      { $$=$2; }
      | expr '+' expr
      { $$=$1+$3; }
      | expr '-' expr
      { $$=$1-$3; }
      | expr '*' expr
      { $$=$1*$3; }
      | expr '/' expr
      { $$=$1/$3; }
      | '-' expr %prec '*'
      { $$=-$2; }
      | NUMBER
      { $$=atoi($1); } ;
%%
```


Yacc specifikacija – sekcija potprograma

```
int yyerror(char *s){
    printf("\n Greska %s u redu %d\n",s,yyline);
}
main(){
    yyline=1;
    if (yyparse()==0) printf("\n Parsiranje uspesno zavrsono\n");
    else printf("\n Parsiranje nije uspesno zavrsono\n");
}
```