YACC

Yet Another Compiler Compiler

Yacc is a parser generator:

Input: A Grammar

Output: A parser for the grammar

(Reminder: a parser finds derivations)

Example grammar:

```
expr -> ( expr )
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
      | - expr
      | INT
      ;
```

The yacc code:

```
expr : '(' expr ')'
     | expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | - expr
     | INT
     ;
```

Exampe Input:

10 * 3 + 4

Yacc Derivation:

expr => expr + expr => expr * expr + expr

=> 10*3 + 4

Resolving Ambiguities

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

expr : '(' expr ')'
     | expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | '-' expr          %prec UMINUS
     | INT
     ;
```

Actions

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

expr : '(' expr ')'          {$$ = $2;}
     | expr '+' expr         {$$ = $1 + $3;}
     | expr '-' expr         {$$ = $1 - $3;}
     | expr '*' expr         {$$ = $1 * $3;}
     | expr '/' expr         {$$ = $1 / $3;}
     | '-' expr        %prec UMINUS    {$$ = -$2;}
     | INT                    {$$ = $1;}
     ;
```

A Complete Yacc program

```
%union{
  int int_val;
}

%left '+', '-'
%left '*', '/'
%left UMINUS

%token <int_val> INT
%type <int_val> expr

%start program

%%
```

```
program : expr        {printf("Expr value = %d \n", $1);}
        | error       {printf("YACC: syntax error near line %d \n", linenum);
                       abort();}
        ;

expr : '(' expr ')'     {$$ = $2;}
     | expr '+' expr    {$$ = $1 + $3;}
     | expr '-' expr    {$$ = $1 - $3;}
     | expr '*' expr    {$$ = $1 * $3;}
     | expr '/' expr    {$$ = $1 / $3;}
     | '-' expr         %prec UMINUS  {$$ = -$2;}
     | INT              {$$ = $1;}
     ;

%%

#include "lex.yy.c"
```

Execution Example

Input: 10 + 20*(3 - 4 + 25)

Output: Expr value = 490

The Lex Code

```
%{
int linenum=1;
int temp_int;
%}
%%

\n     {linenum++;}

[\t ]         /* skip spaces */;
\/\/[^\n]*    /* ignore comments */;

"+"     {return '+';}
"-"     {return '-';}
"*"     {return '*';}
"/"     {return '/';}
")"     {return ')';}
"("     {return '(';}
```

```
[0-9]+   {sscanf(yytext, "%d", &temp_int);
         yylval.int_val = temp_int;
         return INT;}

.  {printf("LEX: unknown input string found in line %d \n", linenum);
    abort();}
```

Compiling:

```
yacc YaccFile
lex LexFile
cc y.tab.c -ly -ll -o MyParser
```

Executable: MyParser

Another Yacc Program

```
%union{
  int int_val;
}

%left '+', '-'
%left '*', '/'
%left UMINUS

%token <int_val> INT
%type <int_val> expr

%start program

%%
```

```
program : stmt_list
        | error         {printf("YACC: syntax error near line %d \n", linenum);
                        abort();}
        ;

stmt_list : stmt_list stmt
          | stmt
          ;

stmt : expr ';'     {printf("Expr value = %d \n", $1);}
     ;
```

```
expr : '(' expr ')'      {$$ = $2;}
     | expr '+' expr     {$$ = $1 + $3;}
     | expr '-' expr     {$$ = $1 - $3;}
     | expr '*' expr     {$$ = $1 * $3;}
     | expr '/' expr     {$$ = $1 / $3;}
     | '-' expr          %prec UMINUS {$$ = -$2;}
     | INT               {$$ = $1;}
     ;

%%

#include "lex.yy.c"
```

Execution Example

Input:

```
10 + 20*(30 -67) / 4;

34 * 35 - 123 + -001;

17*8/6;
```

Output:

```
Expr value = -175
Expr value = 1066
Expr value = 22
```

Lex Code

```
%{
int linenum=1;
int temp_int;
%}
%%

\n     {linenum++;}

[\t ]        /* skip spaces */;
\/\/[^\n]*   /* ignore comments */;
```

```
"+"    {return '+';}
"-"    {return '-';}
"*"    {return '*';}
"/"    {return '/';}
")"    {return ')';}
"("    {return '(';}
";"    {return ';';}

[0-9]+   {sscanf(yytext, "%d", &temp_int);
          yylval.int_val = temp_int;
          return INT;}

.  {printf("LEX: unknown input string found in line %d \n", linenum);
    abort();}
```

Another Yacc Program

```
%union{
  int int_val;
  char *str_val;
}

%left '+', '-'
%left '*', '/'
%left UMINUS

%token PRINT
%token NEWLINE
%token <str_val> STRING
%token <int_val> INT
%type <int_val> expr

%start program
%%
```

```
program : stmt_list
        | error       {printf("YACC: syntax error near line %d \n", linenum);
                       abort();}
        ;

stmt_list : stmt_list stmt
          | stmt
          ;

stmt : expr ';'                  {printf("expression found\n");}
     | PRINT expr ';'            {printf("%d", $2);}
     | PRINT STRING ';'          {printf("%s", $2);}
     | PRINT NEWLINE ';'         {printf("\n");}
     ;
```

```
expr : '(' expr ')'     {$$ = $2;}
     | expr '+' expr    {$$ = $1 + $3;}
     | expr '-' expr    {$$ = $1 - $3;}
     | expr '*' expr    {$$ = $1 * $3;}
     | expr '/' expr    {$$ = $1 / $3;}
     | '-' expr         %prec UMINUS {$$ = -$2;}
     | INT              {$$ = $1;}
     ;

%%

#include "lex.yy.c"
```

Execution Example

Input:

```
print "The value of expression 123 * 25 is ";
print 123 * 25;
print newline;
10 + 5 * 8;
print "end of program";
print newline;
```

Output:

```
The value of expression 123 * 25 is 3075
expression found
end of program
```

Lex Code

```
%{
int linenum=1;
int temp_int;
char temp_str[200];
%}
%%

\n     {linenum++;}

[\t ]          /* skip spaces */;
\/\/[^\n]*     /* ignore comments */;
```

```
"+"         {return '+';}
"-"         {return '-';}
"*"         {return '*';}
"/"         {return '/';}
")"         {return ')';}
"("         {return '(';}
";"         {return ';';}
"print"     {return PRINT;}
"newline"   {return NEWLINE;}
```

```
[0-9]+   {sscanf(yytext, "%d", &temp_int);
          yylval.int_val = temp_int;
          return INT;}

\"[^"\n]*\" {strncpy(temp_str, &(yytext[1]), strlen(yytext)-2);
             temp_str[strlen(yytext)-2] = (char) 0;
              yylval.str_val = temp_str;
              return STRING;}

.  {printf("LEX: unknown input string found in line %d \n", linenum);
    abort();}
```