```
%{
#include <stdio.h>
#include <string.h>

#include "parser.tab.h"
#define YY_USER_ACTION yylloc.first_line = yylloc.last_line = yylineno;
%}

%option yylineno
%option noinput
%option nounput

%%

[ \t\n\r]+

"*"     return '*';
"/"     return '/';
"+"     return '+';
"-"     return '-';
"("     return '(';
")"     return ')';

0|([1-9][0-9]*) {
        yylval.int_val = atoi(yytext);
        return tINTVAL;
}

[a-zA-Z_][a-zA-Z0-9_]* {
        yylval.string_val = strdup(yytext);
        return tIDENTIFIER;
}

.       { fprintf(stderr, "Error: (line %d) unexpected character '%s'\n", yylineno, yytext); exit(1);
}

%%
```

```
%{
#include <stdio.h>
#include <stdlib.h>

#include "tree.h"

extern int yylineno;
extern EXP *root;

int yylex();
void yyerror(const char *s) { fprintf(stderr, "Error: (line %d) %s\n", yylineno, s); exit(1); }
%}

%locations
%error-verbose

%code requires
{
        #include "tree.h"
}

%union {
        int int_val;
        char *string_val;
        EXP *exp;
}

%type <exp> program exp

%token <int_val> tINTVAL
%token <string_val> tIDENTIFIER

%left '+' '-'
%left '*' '/'

%start program

%%
program: exp { root = $1; }
       ;

exp : tIDENTIFIER { $$ = makeEXP_identifier($1); }
    | tINTVAL     { $$ = makeEXP_intLiteral($1); }
    | exp '*' exp { $$ = makeEXP_binary(k_expressionKindMultiplication, $1, $3); }
    | exp '/' exp { $$ = makeEXP_binary(k_expressionKindDivision, $1, $3); }
    | exp '+' exp { $$ = makeEXP_binary(k_expressionKindAddition, $1, $3); }
    | exp '-' exp { $$ = makeEXP_binary(k_expressionKindSubtraction, $1, $3); }
    | '(' exp ')' { $$ = $2; }
    ;

%%
```

```
Package tiny;

Helpers
  tab   = 9;
  cr    = 13;
  lf    = 10;
  digit = ['0'..'9'];
  lowercase = ['a'..'z'];
  uppercase = ['A'..'Z'];
  letter  = lowercase | uppercase;
  idletter = letter | '_';
  idchar  = letter | '_' | digit;

Tokens
  eol   = cr | lf | cr lf;
  blank = ' ' | tab;
  star  = '*';
  slash = '/';
  plus  = '+';
  minus = '-';
  l_par = '(';
  r_par = ')';
  number  = '0'| [digit-'0'] digit*;
  id    = idletter idchar*;

Ignored Tokens
  blank, eol;

Productions
cst_exp {-> exp} =
  {cst_plus}    cst_exp plus factor
                {-> New exp.plus(cst_exp.exp,factor.exp)}
| {cst_minus}   cst_exp minus factor
                      {-> New exp.minus(cst_exp.exp,factor.exp)}
| {factor}      factor {-> factor.exp};

factor {-> exp} =
  {cst_mult}    factor star term
                {-> New exp.mult(factor.exp,term.exp)}
| {cst_divd}    factor slash term
                {-> New exp.divd(factor.exp,term.exp)}
| {term}        term {-> term.exp};

term {-> exp} =
  {paren}       l_par cst_exp r_par {-> cst_exp.exp}
| {cst_id}      id {-> New exp.id(id)}
| {cst_number}  number {-> New exp.number(number)};

Abstract Syntax Tree
exp =
  {plus}     [l]:exp [r]:exp
| {minus}    [l]:exp [r]:exp
| {mult}     [l]:exp [r]:exp
| {divd}     [l]:exp [r]:exp
| {id}       id
| {number}   number;
```