```
%{
#include "y.tab.h"
#include <string.h>
#include <stdlib.h>

extern int lineno;
%}

%%
[ \t]+    /* ignore */;
\n        lineno++;

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

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

[a-zA-Z_][a-zA-Z0-9_]* {
  yylval.stringconst =
    (char *) malloc (strlen (yytext) + 1);
  sprintf (yylval.stringconst, "%s", yytext);
  return tIDENTIFIER;
}

.         /* ignore */;
%%
```

```
%{
#include <stdio.h>
#include "tree.h"

extern char *yytext;
extern EXP *theexpression;

void yyerror() {
   printf ("syntax error before %s\n", yytext);
}
%}

%union {
   int intconst;
   char *stringconst;
   struct EXP *exp;
}

%token <intconst> tINTCONST
%token <stringconst> tIDENTIFIER

%type <exp> program exp

%start program

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

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

exp : tIDENTIFIER
      { $$ = makeEXPid ($1); }
    | tINTCONST
      { $$ = makeEXPintconst ($1); }
    | exp '*' exp
      { $$ = makeEXPtimes ($1, $3); }
    | exp '/' exp
      { $$ = makeEXPdiv ($1, $3); }
    | exp '+' exp
      { $$ = makeEXPplus ($1, $3); }
    | exp '-' exp
      { $$ = makeEXPminus ($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;
```