```
/* mycalc.y   mycalc based on mfcalc present in the bison manual
-*- c -*- */
/*   Time-stamp: <97/09/11 16:59:22 - by alair at tutu.ime.usp.br
>   */

%{
  #include <math.h>  /* For math functions, cos(), sin(), etc. */
  #include "mysymrec.h"  /* Contains definition of `symrec'    */
  double last_print=0;
%}

%union {
  double     val;    /* For returning numbers.                  */
  symrec   *tptr;    /* For returning symbol-table pointers     */
}

%token <val>  NUM '$'     /* Simple double precision number   */
%token <tptr> VAR FNCT    /* Variable and Function             */
%type  <val>  exp

%right '='
%left '-' '+'
%left '*' '/'
%left NEG      /* Negation--unary minus */
%right '^'     /* Exponentiation        */

/* Grammar follows */

%%
input:    /* empty */
        | input line
;

line:
          '\n'
        | exp '\n'   { printf ("\t%.10g\n", last_print=$1); }
        | error '\n' { yyerrok;                  }
;

exp:      NUM                { $$ = $1;                    }
        | '$'                { $$ = last_print;            }
        | VAR                { $$ = $1->value.var;         }
        | VAR '=' exp        { $$ = $3; $1->value.var = $3; }
```

```
        | FNCT '(' exp ')'    { $$ = (*($1->value.fnctptr))($3);}
        | exp '+' exp         { $$ = $1 + $3;                    }
        | exp '-' exp         { $$ = $1 - $3;                    }
        | exp '*' exp         { $$ = $1 * $3;                    }
        | exp '/' exp         { $$ = $1 / $3;                    }
        | '-' exp  %prec NEG  { $$ = -$2;                        }
        | exp '^' exp         { $$ = pow ($1, $3);               }
        | '(' exp ')'         { $$ = $2;                         }
;
/* End of grammar */
%%
#include <stdio.h>

main (int argc, char *argv[] )
{
  create_sym_table (511);
  init_table ();
  yyparse ();
  if ( argc > 1 && !strcmp(argv[1],"-s") ) {
    printf("\nTabela de Simbolos\n");
    print_sym_table ();
  }
}

yyerror (s)  /* Called by yyparse on error */
     char *s;
{
  printf ("%s\n", s);
}

struct init
{
  char *fname;
  double (*fnct)();
};

struct init arith_fncts[]
  = {
      "sin", sin,
      "cos", cos,
      "atan", atan,
      "ln", log,
      "exp", exp,
```

```
      "sqrt", sqrt,
      0, 0
    };

init_table ()  /* puts arithmetic functions in table. */
{
  int i;
  symrec *ptr;
  for (i = 0; arith_fncts[i].fname != 0; i++)
    {
      ptr = putsym (arith_fncts[i].fname, FNCT);
      ptr->value.fnctptr = arith_fncts[i].fnct;
    }
}
```