```
/* multi function calculator -*- c -*- */
/*   Time-stamp: <mfcalc.y (1997/09/10, 19:21:51) by alair at ime
.usp.br)>    */

%{
#include <math.h>  /* For math functions, cos(), sin(), etc. */
#include "mfcalcs.h"  /* Contains definition of `symrec'        *
/
%}
%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", $1); }
        | error '\n' { yyerrok;                  }
;

exp:      NUM                { $$ = $1;                         }
        | VAR                { $$ = $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 */
%%


/*   The new version of `main' includes a call to `init_table', a
function that initializes the symbol table.  Here it is, and
`init_table' as well:
*/
#include <stdio.h>

main ()
{
  init_table ();
  yyparse ();
}

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
```

```
    };

/* The symbol table: a chain of `struct symrec'.  */
symrec *sym_table = (symrec *)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;
    }
}

/*   By simply editing the initialization list and adding the nec
essary
include files, you can add additional functions to the calculator
.

  Two important functions allow look-up and installation of symbo
ls in
the symbol table.  The function `putsym' is passed a name and the
 type
(`VAR' or `FNCT') of the object to be installed.  The object is l
inked
to the front of the list, and a pointer to the object is returned
.  The
function `getsym' is passed the name of the symbol to look up.  I
f
found, a pointer to that symbol is returned; otherwise zero is re
turned.
*/
symrec *
putsym (sym_name,sym_type)
     char *sym_name;
     int sym_type;
{
  symrec *ptr;
  ptr = (symrec *) malloc (sizeof (symrec));
  ptr->name = (char *) malloc (strlen (sym_name) + 1);
  strcpy (ptr->name,sym_name);
```

```
  ptr->type = sym_type;
  ptr->value.var = 0; /* set value to 0 even if fctn.  */
  ptr->next = (struct symrec *)sym_table;
  sym_table = ptr;
  return ptr;
}

symrec *
getsym (sym_name)
     char *sym_name;
{
  symrec *ptr;
  for (ptr = sym_table; ptr != (symrec *) 0;
       ptr = (symrec *)ptr->next)
    if (strcmp (ptr->name,sym_name) == 0)
      return ptr;
  return 0;
}

/*   The function `yylex' must now recognize variables, numeric v
alues,
and the single-character arithmetic operators.  Strings of alphan
umeric
characters with a leading nondigit are recognized as either varia
bles or
functions depending on what the symbol table says about them.

   The string is passed to `getsym' for look up in the symbol tab
le.  If
the name appears in the table, a pointer to its location and its
type
(`VAR' or `FNCT') is returned to `yyparse'.  If it is not already
 in
the table, then it is installed as a `VAR' using `putsym'.  Again
, a
pointer and its type (which must be `VAR') is returned to `yypars
e'.

   No change is needed in the handling of numeric values and arit
hmetic
operators in `yylex'.
*/
     #include <ctype.h>
```

```
yylex ()
{
  int c;

  /* Ignore whitespace, get first nonwhite character.  */
  while ((c = getchar ()) == ' ' || c == '\t');

  if (c == EOF)
    return 0;

/* Char starts a number => parse the number.         */
  if (c == '.' || isdigit (c))
    {
      ungetc (c, stdin);
      scanf ("%lf", &yylval.val);
      return NUM;
    }

/* Char starts an identifier => read the name.       */
  if (isalpha (c))
    {
      symrec *s;
      static char *symbuf = 0;
      static int length = 0;
      int i;

/* Initially make the buffer long enough
         for a 40-character symbol name.  */
      if (length == 0)
        length = 40, symbuf = (char *)malloc (length + 1);

      i = 0;
      do

        {
          /* If buffer is full, make it bigger.        */
          if (i == length)
            {
              length *= 2;
              symbuf = (char *)realloc (symbuf, length + 1);
            }
          /* Add this character to the buffer.         */
          symbuf[i++] = c;
```

```
          /* Get another character.                    */
          c = getchar ();
        }

      while (c != EOF && isalnum (c));

      ungetc (c, stdin);
      symbuf[i] = '\0';

      s = getsym (symbuf);
      if (s == 0)
        s = putsym (symbuf, VAR);
      yylval.tptr = s;
      return s->type;
    }

  /* Any other character is a token by itself.        */
  return c;
}
```