(D0)

The "demo" program is a small interactive calculator. It evaluates arithmetic expressions built from floating point numbers and the four basic arithmetic operations with the usual priorities and associativities. Parentheses allow grouping of subexpressions. There is a global memory with 26 variables named 'a' through 'z'. Assignments are written *after* the expression with the assigment operator '->'. Here is an example: "(1.1+2.2)/(3.3-4.4) -> a" yields -3 as result and stores this value into the variable a. Multiple assigments are possible: "10/3 -> a -> b -> c" stores 3.3333333 into a, b, and c. The variables may be used in expressions and keep their values indefinitely. They are initialized with 0. The program is terminated by entering CTRL-D.

D1

```
/*
 * demo.c -- demonstration program
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>


double memory[26];


/*****************************************************************/

/* scanner */


#define END_OF_INPUT    0
#define NUMBER          1
#define VAR             2
#define NEWLINE         3
#define PLUS            4
#define MINUS           5
#define STAR            6
#define SLASH           7
#define LPAREN          8
#define RPAREN          9
#define STORE           10


typedef union {
  double value;
  int index;
} SemanticValue;


SemanticValue tokenValue;


int nextToken(void) {
  int c;
  double value, weight;
  int s, e;

  c = getchar();
  while (c == ' ' || c == '\t') {
    c = getchar();
  }
  if (c == EOF) {
    return END_OF_INPUT;
  }
  if (c == '.' || isdigit(c)) {
    value = 0.0;
    while (isdigit(c)) {
      value *= 10.0;
      value += c - '0';
      c = getchar();
    }
    if (c == '.') {
      c = getchar();
      weight = 0.1;
      while (isdigit(c)) {
        value += (c - '0') * weight;
        weight /= 10.0;
        c = getchar();
```

```
      }
    }
    if (c == 'e') {
      c = getchar();
      if (c == '+') {
        s = 1;
        c = getchar();
      } else
      if (c == '-') {
        s = -1;
        c = getchar();
      } else {
        s = 1;
      }
      if (!isdigit(c)) {
        printf("no digit after 'e' in number\n");
        exit(99);
      }
      e = 0;
      while (isdigit(c)) {
        e *= 10;
        e += c - '0';
        c = getchar();
      }
      value *= pow(10, s * e);
    }
    ungetc(c, stdin);
    tokenValue.value = value;
    return NUMBER;
  }
  if (islower(c)) {
    tokenValue.index = c - 'a';
    return VAR;
  }
  if (c == '\n') {
    return NEWLINE;
  }
  if (c == '+') {
    return PLUS;
  }
  if (c == '-') {
    c = getchar();
    if (c == '>') {
      return STORE;
    }
    ungetc(c, stdin);
    return MINUS;
  }
  if (c == '*') {
    return STAR;
  }
  if (c == '/') {
    return SLASH;
  }
  if (c == '(') {
    return LPAREN;
  }
  if (c == ')') {
    return RPAREN;
  }
  printf("illegal character '%c' (0x%02X)\n", c, c);
  exit(99);
  /* not reached */
  return 0;
}

/*************************************************************/
```

```
/* parser */


void parseList(void);
SemanticValue parseAssign(void);
SemanticValue parseExpr(void);
SemanticValue parseTerm(void);
SemanticValue parseFactor(void);
SemanticValue parsePrimary(void);


int lookahead;


void syntaxError(char *msg) {
  printf("syntax error: %s\n", msg);
  exit(99);
}


void parseList(void) {
  SemanticValue value;

  lookahead = nextToken();
  while (lookahead != END_OF_INPUT) {
    if (lookahead == NEWLINE) {
      lookahead = nextToken();
    } else {
      value = parseAssign();
      if (lookahead != NEWLINE) {
        syntaxError("newline expected");
      }
      printf("\t%.8g\n", value.value);
    }
  }
}


SemanticValue parseAssign(void) {
  SemanticValue value;

  value = parseExpr();
  while (lookahead == STORE) {
    lookahead = nextToken();
    if (lookahead != VAR) {
      syntaxError("variable expected");
    }
    memory[tokenValue.index] = value.value;
    lookahead = nextToken();
  }
  return value;
}


SemanticValue parseExpr(void) {
  SemanticValue value;
  SemanticValue aux;

  value = parseTerm();
  while (lookahead == PLUS || lookahead == MINUS) {
    if (lookahead == PLUS) {
      lookahead = nextToken();
      aux = parseTerm();
      value.value += aux.value;
    } else
    if (lookahead == MINUS) {
      lookahead = nextToken();
```

```
      aux = parseTerm();
      value.value -= aux.value;
    }
  }
  return value;
}


SemanticValue parseTerm(void) {
  SemanticValue value;
  SemanticValue aux;

  value = parseFactor();
  while (lookahead == STAR || lookahead == SLASH) {
    if (lookahead == STAR) {
      lookahead = nextToken();
      aux = parseFactor();
      value.value *= aux.value;
    } else
    if (lookahead == SLASH) {
      lookahead = nextToken();
      aux = parseFactor();
      if (aux.value == 0.0) {
        printf("division by zero\n");
        exit(99);
      }
      value.value /= aux.value;
    }
  }
  return value;
}


SemanticValue parseFactor(void) {
  SemanticValue value;

  if (lookahead == PLUS) {
    lookahead = nextToken();
    value = parseFactor();
    return value;
  }
  if (lookahead == MINUS) {
    lookahead = nextToken();
    value = parseFactor();
    value.value = -value.value;
    return value;
  }
  value = parsePrimary();
  return value;
}


SemanticValue parsePrimary(void) {
  SemanticValue value;

  if (lookahead == NUMBER) {
    value.value = tokenValue.value;
    lookahead = nextToken();
    return value;
  }
  if (lookahead == VAR) {
    value.value = memory[tokenValue.index];
    lookahead = nextToken();
    return value;
  }
  if (lookahead == LPAREN) {
    lookahead = nextToken();
    value = parseExpr();
```

```
    if (lookahead != RPAREN) {
      syntaxError("right parenthesis expected");
    }
    lookahead = nextToken();
    return value;
  }
  syntaxError("number, variable, or left parenthesis expected");
  /* never reached */
  value.value = 0.0;
  return value;
}


/*****************************************************************/

/* driver */


int main(int argc, char *argv[]) {
  int i;

  printf("Demo started:\n");
  for (i = 0; i < 26; i++) {
    memory[i] = 0.0;
  }
  parseList();
  printf("Demo stopped.\n");
  return 0;
}
```