Input File:

```
D      5.20    -     31.05
D    17.462    /         4
I       776    -       612
A       3.4    *         2
I        16    +       113
D     8.720    +      12.5
D         7    /         4
I         7    x         4
E
I         1    +         1
```

Output File:

```
Simple Calculator by
                                Bill McQuain

          Expression                    Value
-----------------------------------------------
      5.20    -     31.05    =    -25.85
-----------------------------------------------
     17.46    /      4.00    =      4.37
-----------------------------------------------
       776    -       612    =       164
-----------------------------------------------
Bad expression code
-----------------------------------------------
        16    +       113    =       129
-----------------------------------------------
      8.72    +     12.50    =     21.22
-----------------------------------------------
      7.00    /      4.00    =      1.75
-----------------------------------------------
Bad operator:  x
-----------------------------------------------
```

```
// CS 1044 Example Fall 1998
//
// Simple 4-function calculator for integer and real expressions.
//
#include <fstream.h>
#include <iomanip.h>

//  Enumerated type used to recall type of expression flag:
enum DataType {Integer, Real, Unknown, Finished};

////////////////////////////////////////////////////////////////
//  File management functions:
bool   SetFiles(ifstream& iFile, ofstream& oFile);
void CloseFiles(ifstream& iFile, ofstream& oFile);

//  Input flag remapping
DataType ClassifyExpression(char EquationType);

//  Driver function for equation processing
void   ProcessEquation(ifstream& iFile, ofstream& oFile, DataType EquationType);

//  Expression evaluation functions:
int    DoIntegerOperation(int Operand1, char Operator, int Operand2);
double DoRealOperation(double Operand1, char Operator, double Operand2);

//  Expression validation functions:
bool   CheckOperator(char Operator);

//  I/O functions:
void   PrintHeader(ofstream& oFile);
void   PrintIntegerResults(ofstream& oFile, int Operand1, char Operator,
                           int Operand2, int Result);
void   PrintRealResults(ofstream& oFile, double Operand1, char Operator,
                        double Operand2, double Result);
```

```
////////////////////////////////////////////////////////////////////////
//  Begin main program:
void main() {

   char ExpressionType;       // flag indicating how data line is
                              //     to be processed (I, D, E)
   DataType EquationType;     // flag indicating type of equation
                              //     to be processed

   ifstream iFile;            // input file stream
   ofstream oFile;            // output file stream

   // Try to open and set up input and output files:
   if (!SetFiles(iFile, oFile)) {
      cout << "Error setting up files... exiting." << endl;
      return;
   }

   //  Write header to output file:
   PrintHeader(oFile);

   //  Try to read expression type for first line of input:
   iFile.get(ExpressionType);
   EquationType = ClassifyExpression(ExpressionType);
```

```
   //  Continue processing until the exit code 'E' or EOF
   //  is encountered:
   while (EquationType != Finished && iFile) {

      // The equation type determines how to read the rest
      // of the input line and how to process the values found:
      ProcessEquation(iFile, oFile, EquationType);

      // Skip to beginning of next input line:
      iFile.ignore(80, '\n');

      // Read next expression type:
      iFile.get(ExpressionType);
      EquationType = ClassifyExpression(ExpressionType);
   }

   // Close file streams:
   CloseFiles(iFile, oFile);
}
```

```
//////////////////////////////////////////////////////////////////////
//  SetFiles attempts to open the input and output files and prepare the
//  output stream for floating-point output.
//
//  Parameters:
//     iFile    input file stream
//     oFile    output file stream
//  Returns:
//     true     if setup succeeds
//     false    otherwise
//
//  Pre:  input file should exist
//  Post: input and output files are open, output stream prepared for
//        floating-point output
//
bool SetFiles(ifstream& iFile, ofstream& oFile) {

   iFile.open("calc.dat", ios::nocreate); // try to open input file
   if (iFile.fail())
      return false;
   oFile.open("calc.out");                // open output file
   oFile.setf(ios::fixed, ios::floatfield); oFile.setf(ios::showpoint);
   return true;
}
```

```
////////////////////////////////////////////////////////////////////////
//  CloseFiles closes the input and output files.
//
//  Parameters:
//     iFile    input file stream
//     oFile    output file stream
//  Returns:
//     nothing
//
//  Pre:  nothing
//  Post: input and output files are closed
//
void CloseFiles(ifstream& iFile, ofstream& oFile) {

   iFile.close();
   oFile.close();
}
```

```
//////////////////////////////////////////////////////////////////////
//  ClassifyExpression returns a DataType value indicating the kind of
//  expression to be parsed.
//
//  Parameter:
//     EquationType  char flag from input file
//  Returns:
//     Integer       if EquationType is 'I'
//     Real          if EquationType is 'D'
//     Finished      if EquationType is 'E'
//     Unknown       if EquationType is anything else
//
//  Pre:  EquationType has been initialized
//  Post: return value corresponds to significance of EquationType
//
DataType ClassifyExpression(char EquationType) {

   DataType KindOfExpression;

   switch (EquationType) {
   case 'I': KindOfExpression = Integer;
             break;
   case 'D': KindOfExpression = Real;
             break;
   case 'E': KindOfExpression = Finished;
             break;
   default : KindOfExpression = Unknown;
   }

   return KindOfExpression;
}
```

```
////////////////////////////////////////////////////////////////////////
//  ProcessEquation manages the handline of the current equation line.
//
//  Parameter:
//     Operator  the operator in question
//  Returns:
//     true    if operator is +, -, *, or /
//     false   otherwise
//
//  Pre:  files are open; EquationType has been initialized
//  Post: corresponding expression has been read and evaluated correctly,
//        and expression and value have been written to output file
//
void ProcessEquation(ifstream& iFile, ofstream& oFile, DataType EquationType) {

   char Operator;         // numeric operator (+, -, *, /)

                          // These are used for integer expressions:
   int  iOperand1,        //      first operand
        iOperand2,        //      second operand
        iResult;          //      computed result
                          // These are used for double expressions:
   double dOperand1,      //      first operand
          dOperand2,      //      second operand
          dResult;        //      computed result
```

```
switch (EquationType) {
case Integer: {// Read integer expression:
               iFile >> iOperand1 >> Operator >> iOperand2;
               // Check for valid operator:
               if (CheckOperator(Operator)) {
                  // If valid, compute integer result and print to output file:
                  iResult = DoIntegerOperation(iOperand1, Operator, iOperand2);
                  PrintIntegerResults(oFile, iOperand1, Operator, iOperand2,
                                      iResult);
               }
               else
                  // If not valid, print error message:
                  oFile << "Bad operator:  " << Operator << endl
                        << "---------------------------------------" << endl;
               break;
               }
case Real:     {  // Read double expression:
               iFile >> dOperand1 >> Operator >> dOperand2;
               // Check for valid operator:
               if (CheckOperator(Operator)) {
                  // If valid, compute double result and print to output file:
                  dResult = DoRealOperation(dOperand1, Operator, dOperand2);
                  PrintRealResults(oFile, dOperand1, Operator, dOperand2,
                                   dResult);
               }
               else
                  // If invalid, print error message:
                  oFile << "Bad operator:  " << Operator << endl
                        << "---------------------------------------" << endl;
               break;
               }
```

```
   case Finished:{// If the exit code was just read, return.
                 // (This case should never, logically, occur.)
                 return;
                 }
   default:      {  // If expression code is bad, output error message:
                 oFile << "Bad expression code" << endl
                       << "---------------------------------------" <<endl;
                 }
   };
}
```

```
//////////////////////////////////////////////////////////////////////////
//  CheckOperator determines if the specified operator is valid.
//
//  Parameter:
//     Operator  the operator in question
//  Returns:
//     true    if operator is +, -, *, or /
//     false   otherwise
//
//  Pre:  Operator has been initialized
//  Post: return value indicates if Operator is a valid arithmetic symbol
//
bool CheckOperator(char Operator) {

   if (Operator == '+' || Operator == '-' || Operator == '*' || Operator == '/')
      return true;
   else
      return false;
}
```

```
////////////////////////////////////////////////////////////////////////
//  DoIntegerOperation applies an integer operator to two integer operands.
//
//  Parameters:
//     Operand1     first operand
//     Operator     operator
//     Operand2     second operand
//  Returns:
//     Result  integer value of expression (Op1 Op Op2)
//
//  Pre:  Operand1 and Operand2 have been initialized; Operator is valid
//  Post: return value equals result of performing specified calculation
//
int DoIntegerOperation(int Operand1, char Operator, int Operand2) {
   int Result;

   switch (Operator) {
   case '+': Result = Operand1 + Operand2;
             break;
   case '-': Result = Operand1 - Operand2;
             break;
   case '*': Result = Operand1 * Operand2;
             break;
   case '/': Result = Operand1 / Operand2;
             break;
   default:  Result = 0;
   }
   return Result;
}
```

```
////////////////////////////////////////////////////////////////////////
//  DoRealOperation appliess a double operator to two double operands.
//
//  Parameters:
//     Operand1     first operand
//     Operator     operator
//     Operand2     second operand
//  Returns:
//     Result  double value of expression (Op1 Op Op2)
//
//  Pre:  Operand1 and Operand2 have been initialized; Operator is valid
//  Post: return value equals result of performing specified calculation
//
double DoRealOperation(double Operand1, char Operator, double Operand2) {
   double Result;

   switch (Operator) {
   case '+': Result = Operand1 + Operand2;
             break;
   case '-': Result = Operand1 - Operand2;
             break;
   case '*': Result = Operand1 * Operand2;
             break;
   case '/': Result = Operand1 / Operand2;
             break;
   default:  Result = 0;
   }
   return Result;
}
```

```
////////////////////////////////////////////////////////////////////////
//  PrintHeader prints the specified header to the output file.
//
//  Parameters:
//     oFile        output file stream
//  Returns:
//     nothing
//
//  Pre:  output file has been opened
//  Post: file header has been written
//
void PrintHeader(ofstream& oFile) {

   oFile << "Simple Calculator by" << endl;
   oFile << "                              Bill McQuain" << endl;
   oFile << endl;
   oFile << "          Expression               Value" << endl;
   oFile << "-----------------------------------------" << endl;
}
```

```
//////////////////////////////////////////////////////////////////////////
//  PrintIntegerResults prints the given integer expression and its computed value
//  to the output file stream.
//
//  Parameters:
//     oFile        output file stream
//     Operand1     first operand
//     Operator     operator
//     Operand2     second operand
//  Returns:
//     nothing
//
//  Pre:  output file has been opened; other parameters have been initialized
//  Post: expression and value have been written to output file
//
void PrintIntegerResults(ofstream& oFile, int Operand1, char Operator, int Operand2,
int Result) {

   oFile << setw(10) << Operand1
         << setw( 5) << Operator
         << setw(10) << Operand2
         << setw( 5) << '='
         << setw(10) << Result
         << endl;
   oFile << "----------------------------------------" << endl;
}
```

```
//////////////////////////////////////////////////////////////////////////
//  PrintRealResults prints the given double expression and its computed value
//  to the output file stream.
//
//  Parameters:
//     oFile        output file stream
//     Operand1     first operand
//     Operator     operator
//     Operand2     second operand
//  Returns:
//     nothing
//
//  Pre:  output file has been opened; other parameters have been initialized
//  Post: expression and value have been written to output file
//
void PrintRealResults(ofstream& oFile, double Operand1, char Operator, double
Operand2, double Result){

   oFile << setw(10) << setprecision(2) << Operand1
         << setw( 5) << Operator
         << setw(10) << setprecision(2) << Operand2
         << setw( 5) << '='
         << setw(10) << setprecision(2) << Result
         << endl;
   oFile << "----------------------------------------" << endl;
}
```