```c
/*
 * Remember the Unix Philosophy:
 * “Write programs that do one thing and do it well. Write
 * programs to work together. Write programs to handle
 * text streams, because that is a universal interface”
 * - Doug McIlroy
 */

typedef short int CALC_ERROR;

typedef struct Calculator
{
  Stack sStk;
  /* possibly other data */
} Calculator;

/*
 * create and terminate the calculator, in case there is any dynamic
 * memory or state to setup/tear-down
 */
CALC_ERROR calcCreateCalculator(Calculator*);
CALC_ERROR calcTerminateCalculator(Calculator*);

/*
 * The calculator will handle a text stream of data.
 * The user needs to indicate when a new stream of data begins and
 * and when the stream ends.  When a stream ends the calculator will
 * parse the stream and produce either an error message or the final result.
 *
 * We want the calculator to parse the stream of data so that we do not
 * depend on the driver (one driver of many) to know how to parse the
 * data.
 *
 * If we want to expand the calculator beyond the restrictions of this
 * simple assignment (single digit values, add symbolic values), we don't
 * want every driver to need to change.
 *
 * The calculator, probably through a parser module hidden from the
 * user, should parse the data stream.
 */
CALC_ERROR calcStartExpression(Calculator*);
CALC_ERROR calcEndExpression(Calculator*);

/*
 * The user can insert either a single character or chunks of
 * characters (strings) into the stream.
 */
CALC_ERROR calcInsertChar(Calculator*, char);
CALC_ERROR calcInsertString(Calculator *, char*);

/*
 * retreive the final result
 */
CALC_ERROR calcGetResult(Calculator*, double*);
```

```c

int main()
{
  Calculator sCalc;
  FILE *pFile;
  char fileData;
  double digit;
  BOOLEAN bExprOpen = FALSE;

  pFile = fopen("testfiles/testdata.txt","r");

  if( NULL == pFile)
  {
    printf("Cannot open file!\n");
    return -1;
  }

  calccreateCalculator(&sCalc);

  while ( EOF != (fileData = fgetc(pFile) ) )
  {
    if( '\n' == fileData )
    {
      calcEndExpression(&sCalc);
      bExprOpen = FALSE;
      calcGetResult(&sCalc, &digit);

      printf("%g\n", digit);
    }
    else
    {
      if( !bExprOpen )
      {
        calcStartExpression(&sCalc);
        bExprOpen = TRUE;
      }
      calcInsertChar(&sCalc, fileData);
    }
  }

  if( bExprOpen )
  {
    calcEndExpression(&sCalc);
    calcGetResult(&sCalc, &digit);

    printf("%g\n", digit);
  }

  fclose(pFile);
  calcTerminateCalculator(&sCalc);

  return 0;
}

```