Program Calculator

Input: Reads an expression from standard input, which is built from natural numbers and * / + operators. The expression ends with the operator =.
Output: The result of the expression as a fraction of two natural numbers.

Originated by Reuven Bar-Yehuda, modified by Y. Kimchi

EXAMPLES:

INPUT: 2 * 3 / 0 + 3 * 5 =
OUTPUT: Fatal error: divisor is 0

INPUT: 23 * 3 / + 5 =
OUTPUT: Fatal error: Illegal character input '+'

INPUT: 2 * 3 / 36/5 / 35=
OUTPUT: Result of calculation is 13/42

#include <stdio.h>
#include <stdlib.h>     /* For exit() etc. */
#include <ctype.h>

void calculate(int *p_numerator, int *p_denominator);

void    /* The result −− Left argument −− Right argument */ multiply(int *pa, int *pb, int a1, int b1, int a2, int b2) {
  reduce(a1, b2); /* Cross reduction prevents premature overflow */
  reduce(a2, b1);
  *pa = a1*a2; /* numerator */
  *pb = b1*b2; /* denominator */
  return; /* Assuming original arguments were reduced, */
}
/* no need for post-reduction */

void    /* The result − Left argument − Right argument */ add(int *pa, int *pb, int a1, int b1, int a2, int b2) {
  *pb = lcm(b1,b2);
  *pa = a1*(pb/b1) + a2*(pb/b2); /* Parentheses control overflow */
  reduce(pa, pb);
  return;
}

void    /* Calculates a long chain of multiplicative operations */
getTermOperator(int *p_a, int *p_b, int *p_operator){
  int a_sav   = 0, b_sav = 1, /* As if every computation begins with "0/1 + ..."  */
  oper    = '+';    /* Previous "oper" was '+' */
  while (oper != '=') {    /* Oper must be '+' or '=' */
    int a_new, b_new;    /* Oper* must be 'r' or '=' */
    getTermOperator(&a_new, &b_new, &oper);
    add(&a_sav, &b_sav, a_sav, b_sav, a_new, b_new);
  }
  /* Send the final result */
  *p_numerator = a_sav;
  *p_denominator = b_sav;
  return;
}

/* Greatest Common Divisor */
int gcd(int a, int b) /* Wrong for gcd(0,0) */{
  if (a < 0) a = −a;
  if (b < 0) b = −b;
  while (b !=0){
    int c = a % b;
    a = b;
    b = c;
  }  return a;
}

/* Least Common Multiplyer */
int lcm(int a, int b) /* DO NOT lcm(0,0) */{
  return a/gcd(a,b)*b; /* Prevents premature overflow */
}

/* DO NOT reduce(0,0) */
void    /* Calculates a long chain of multiplicative operations */
reduce(int *p_numerator, int *p_denominator){
  int cd = gcd(*p_numerator, *p_denominator);
  *p_numerator /= cd;
  *p_denominator /= cd;
return;
}
void fatalErrorTermination(int c) {
    printf("Fatal error: Illegal character input \'%c\n\', c);
    exit(1);
}

int isOperator(int c) {
    return (c == '+' || (c == '*' || (c == '/' || (c == '='));
}

int LegalChar(int c) {
    return isdigit(c) || isOperator(c) || isspace(c));
}

int getChar(void) {
    int c = getchar();
    if (!LegalChar(c)) fatalErrorTermination(c);
    return c;
}

int getFirstNonSpace(void) {
    int c;
    for (c = getChar(); isspace(c); c = getChar());
    return c;
}

int getFirstDigit(void) {
    int c = getFirstNonSpace();
    if (!isdigit(c)) fatalErrorTermination(c);
    return c;
}

#include <stdio.h>

#define TITLE ("Memory Corruption: How easy it is\n")
#define BUF_SIZE (8)

int i;
char c[BUF_SIZE], buf[BUF_SIZE];

main() {
    printf("\#if 0\n");
    printf(TITLE);
    printf("Type string c[]\n");
    (void)gets(c, BUF_SIZE, stdin);
    printf("\%s\n", c);
    printf("Type string buf[]\n");
    (void)gets(buf, BUF_SIZE, stdin);
    printf("\%s\n", buf);
    for (i = 4; i > -4; i--) {
        c[i] = 'a';
        printf("i = \%d, buf =\%s\n", i, buf, c);
    }
    printf("\#endif\n");
    return 0;
} /* End − main() − */

#if 0

Memory Corruption: How easy it is

Type string c[]
012
Type string buf[]
0123456
i = 4, buf = 0123456 c = 012
i = 3, buf = 0123456 c = 012aayý
i = 2, buf = 0123456 c = 01aaayý
i = 1, buf = 0123456 c = 0aaayý
i = 0, buf = 0123456 c = aayaayý
i = -1, buf = 0123456aaaaayý c = aaaaayý
i = -2, buf = 0123456aaaaayý c = aaaaayý
i = -3, buf = 0123456aaaaayý c = aaaaayý
#endif

#include <stdio.h>

#define TITLE ("\n\tMemory Corruption: How easy it is\n")
#define BUF_SIZE (8)

int i;
char c[BUF_SIZE], buf[BUF_SIZE];

main() {
    printf("\#if 0\n");
    printf(TITLE);
    printf("Type string c[]\n");
    (void)gets(c, BUF_SIZE, stdin);
    printf("%s\n", c);
    printf("Type string buf[]\n");
    (void)gets(buf, BUF_SIZE, stdin);
    printf("%s\n", buf);
    for (i = 4; i > -4; i--) {
        c[i] = 'a';
        printf("i = %d, buf = %s c = %s\n", i, buf, c);
    }
    printf("\#endif\n");
    return 0;
} /* End − main() − */

#include <stdio.h>

#define TITLE ("\n\tMemory Corruption: How easy it is\n")
#define BUF_SIZE (8)

int i;
char c[BUF_SIZE], buf[BUF_SIZE];

main() {
    printf("\#if 0\n");
    printf(TITLE);
    printf("Type string c[]\n");
    (void)gets(c, BUF_SIZE, stdin);
    printf("%s\n", c);
    printf("Type string buf[]\n");
    (void)gets(buf, BUF_SIZE, stdin);
    printf("%s\n", buf);
    for (i = 4; i > -4; i--) {
        c[i] = 'a';
        printf("i = %d, buf = %s c = %s\n", i, buf, c);
    }
    printf("\#endif\n");
    return 0;
} /* End − main() − */

#include <stdio.h>

#define TITLE ("\n\tMemory Corruption: How easy it is\n")
#define BUF_SIZE (8)

int i;
char c[BUF_SIZE], buf[BUF_SIZE];

main() {
    printf("\#if 0\n");
    printf(TITLE);
    printf("Type string c[]\n");
    (void)gets(c, BUF_SIZE, stdin);
    printf("%s\n", c);
    printf("Type string buf[]\n");
    (void)gets(buf, BUF_SIZE, stdin);
    printf("%s\n", buf);
    for (i = 4; i > -4; i--) {
        c[i] = 'a';
        printf("i = %d, buf = %s c = %s\n", i, buf, c);
    }
    printf("\#endif\n");
    return 0;
} /* End − main() − */