Homework #7 – Solution
(C Programming for Beginners - OnLine)

Note: *Complete through Lesson#21 (lecture and demo videos) before attempting this homework, requires very basic understanding of pointers, lecture# 13*

Extend the simple calculator you wrote in Midterm, and make it more advanced by making it behave more like how the today's calculator user interface works. For example allow it to accept the input as an equation and print the results out to user.

Details: You need to write a program that when run, will clear the screen and asks the user to write the simple equation they want to evaluate. If the user enters a valid equation, it then displays the result. If user input is not valid, it displays an error message and asks for the correct equation. After the result is shown, it asks the user to enter another equation or press enter without any character to exit the program. The program continues infinitely until the user hits enter without entering anything else.

For simplicity, your program accepts only +, -, * and / as operator. Only one operator is allowed per equation operating on two valid number operands.

Sample Run of the program:

Welcome to <*John Doe*>'s Advanced Calculator, please enter a valid equation followed by enter :
12 + 12.00 ← (<user presses enter>)
12.00 + 12.00 = 24.00

Please enter a valid equation and press enter or press enter to exit.
$5 * 9.0 ← (<user presses enter>)

Not a valid equation.
Please enter a valid equation and press enter or press enter to exit.
2*1 ← (<user presses enter>)
2.00 + 1.00 = 2.00

Please enter a valid equation and press enter or press enter to exit.
5 * 9.0 + 3 ← (<user presses enter>)

Not a valid equation.
Please enter a valid equation and press enter or press enter to exit.
← (<user presses enter>)

Thank you for using <*John Doe*>*'s* Advance Calculator, good bye!
Press enter to exit

Note:

1) Replace the <*John Doe*'> with your name
2) You should allow only four operators (+, -, *, /). And only one operator per equation.
3) Your program should allow input of integer or decimal numbers. The output should always be in decimals with two decimal digits as precision. A comma, or a $ in a number is invalid
4) Give appropriate message when invalid menu choice is selected and re-display theprompt.
5) You should also appropriately catch the divide by zero issue and invalid inputs – e.g. if someone enters characters instead of numbers.
6) When user enters wrong value, give the appropriate message and ask for the correct values again. Repeat until valid values are entered.
7) Program only ends when user presses enter without writing any characters.

Solution:

```
/*
 This program extend the simple calculator.
 It accepts the input as an equation and print the results to user.
 If the user enters a valid equation, it then displays the result.
 If user input is not valid, it displays an error message and asks for the
correct equation.
 After the result is shown, it asks the user to enter another equation or press
enter without any character to exit the program.
 The program continues infinitely until the user hits enter without entering
anything else.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

// Get user input equaton and divide it into three strings: first number,
operatorChar and second number
int getEquation(char *firstNumber, char *secondNumber, char *operatorChar);

// Validate input equation
int validateEquation(char *firstNumber, char *secondNumber, char
*operatorChar);

// Calculte and print result
void calculate(char *firstNumber, char *secondNumber, char *operatorChar);

// Reset all char in string to '\0'
void resetString(char *string);

// Check whether a string can be converted to number
int validateNumber(char *string);

// Check whether input char is a valid number or  a period
```

```c
int validateChar(char *charactor);

void clearScreen();

int main()
{
     // string for first number in equation
    char firstNumber[80]="";
    // string for second number in equation
    char secondNumber[80]="";
    // char for operatorChar in equation. Default value is '0'. Valid value are
    //'+', '-', '*', '/'
    char operatorChar='0';
     clearScreen();
    printf("Welcome to Zeng Wang's Advanced Calculator, please enter a valid
equation followed by enter:\n");
    // program continues infinitely until the user hits enter without entering
    //anything else
    while (getEquation(firstNumber, secondNumber, &operatorChar)) {
        // clear up buffer
        fseek(stdin,0,SEEK_END);
        // if equation is valid, calculator and print result
        if (validateEquation(firstNumber, secondNumber, &operatorChar))
            calculate(firstNumber, secondNumber, &operatorChar);
        printf("Please enter a valid equation and press enter or press enter to
exit.\n");
    }
    // user hit enter without entering anything else. Program exit loop. Wait
    //user to press 'ENTER' to exit.
    printf("Thanks for using Zeng Wang's Advanced Calculator, good Bye!\n");
    printf("Press ENTER to exit\n");
    getchar();
    return 0;
}

void clearScreen(void)
{
     #ifdef linux
       system("clear");
     #else
       system("cls");
#endif
}

// Get user input equaton and divide it into three strings: first number,
//operatorChar and second number
int getEquation(char *firstNumber, char *secondNumber, char *operatorChar)
{
    char input;
    int charIndex = 0;
    // reset strings of first number and second number
    resetString(firstNumber);
    resetString(secondNumber);
    // get user input
    input = getchar();
    // if user hits enter without entering anything else, return to main
    //function
    if (input == '\n')
    {
        return 0;
    }
```

```
// if first input char is '-' or '+', accept it as part of first number.
else if (input == '-' || input == '+')
{
    firstNumber[charIndex]=input;
    charIndex++;
    input=getchar();
}
// get first number and operatorChar
while (input !='\n')
{
    // capture valid operatorChar. Assign chars entered before operatorChar
    //to first number and chars entered after operatorChar to second number
    switch (input)
    {
        case '+':
            *operatorChar = '+';
            // reset charIndex to 0 to catch second number
            charIndex = 0;
            break;
        case '-':
            *operatorChar = '-';
            charIndex = 0;
            break;
        case '*':
            *operatorChar = '*';
            charIndex = 0;
            break;
        case '/':
            *operatorChar = '/';
            charIndex = 0;
            break;
        default:
            if (validateChar(&input)) {
                firstNumber[charIndex] = input;
                charIndex++;
            }
            // if char is invalid, return to main function
            else
                return 1;
    }
    input=getchar();
    // if charIndex is reset to 0, save to second number
    if (charIndex == 0)
        break;
}

// get second number
while (input != '\n')
{
    if (validateChar(&input)) {
        secondNumber[charIndex] = input;
        charIndex++;
    }
    // if char is invalid, reset secondNumber to "" and return to main
    //function
    else
    {
        resetString(secondNumber);
        return 1;
    }
    input = getchar();
```

```
    }
    return 1;
}

// Reset all char in string to '\0'
void resetString(char *string)
{
    // loop through each character of the array and set it to \0
    int i = 0;
    while (string[i])
    {
        // if it is the end of a string, stop
        if (string[i] == '\0')
            break;
        string[i]='\0';
        i++;
    }
}


// Valide whether char is a number or a period or space
int validateChar(char *charactor)
{
    if (*charactor >= 48 && *charactor <= 57)
        return 1;
    else if (*charactor == '.')
        return 1;
     else if (*charactor == ' ')
        return 1;
    return 0;
}

// Validate user input equation
int validateEquation(char *firstNumber, char *secondNumber, char *operatorChar)
{
    // if no valid operatorChar is inputed or first number can't be converted
    //to a valid float or second number can't be converted to a valid float,
    //input equation is invalid
    if (*operatorChar == '0' || secondNumber[0] =='\0'||
!validateNumber(firstNumber) || !validateNumber(secondNumber))
    {
        printf("\nNot a valid equation.\n");
        return 0;
    }
    // if denominator is zero, input equation is invalid
    else if(*operatorChar == '/' && atof(secondNumber) == 0.00)
    {
        printf("\nNot a valid equation. Denominator can't be zero.\n");
        return 0;
    }
    // otherwise, equation is valid.
    else
        return 1;
}

// Use strtoul to validate whether a string can be converted to float
int validateNumber(char *string)
{
    unsigned long int convertedNumber;
    errno = 0;
    convertedNumber=strtoul(string, NULL, 0);
```

```
    // if convertion is failed, it returns error
    if (errno!=0) {
        return 0;
    }
    return 1;
}

// Calculate equation and print result
void calculate(char *firstNumber, char *secondNumber, char *operatorChar)
{
    switch (*operatorChar)
    {
        case '+':
            printf("%.2f + %.2f = %.2f\n\n", atof(firstNumber),
atof(secondNumber), atof(firstNumber) + atof(secondNumber));
            break;
        case '-':
            printf("%.2f - %.2f = %.2f\n\n", atof(firstNumber),
atof(secondNumber), atof(firstNumber) - atof(secondNumber));
            break;
        case '*':
            printf("%.2f * %.2f = %.2f\n\n", atof(firstNumber),
atof(secondNumber), atof(firstNumber) * atof(secondNumber));
            break;
        default:
            printf("%.2f / %.2f = %.2f\n\n", atof(firstNumber),
atof(secondNumber), atof(firstNumber) / atof(secondNumber));
            break;
    }
}
```