CS205 C/C++ Programming – Project 1

Name: Anthony Bryan

SID: 12112230

Part 1 – Analysis

The problem is making a simple calculator that can add, subtract, multiply and divide two numbers using C programming language.

The input of this problem are consists 3 arguments. The first and third argument is number, and the second argument is the operator("+","-","*","/").

The number could be in the form of an integer, decimal, or Scientific number.

How to process the input?

First, I divide the number into two parts. The first part is the number before 'e' and the second part is the number after 'e'. If the number is integer or decimal then we don't have two divide it into two parts.

Second, if the number before 'e' is decimal then store the location of the dot ('.') to help us determine the result after calculation, this also applied to input number is decimal. **If the input number is an integer then the first and second step could be neglected.**

Third if the input number at least one of them is scientific then there exits number after 'e'. So I have to make the number after 'e' between two numbers is equal (if one of the number is not scientific than the number after 'e' is considered as 0 (zero). To make equal the largest number after 'e' minus the smallest number after 'e' and then we got the result (**store the result to e_total**). If the result is negative then the number before 'e' that has largest number after 'e' the dot ('.') will be moved to the left as much as the result number. Else the dot ('.') will be moved to the right as much as the result number. If result is 0 (zero) than don't need to do anything. If the number before e is integer then the dot '.' is start from the most right of the number. **This is applied for the operation of addition and substraction only.**

Calculation part …

Addition

If the two numbers are integers and positive than we can just do an addition like this **(Remove the dot if there any)**.

addtionIntegersFuntion

- Reverse both numbers.
- Do an addition of two numbers element by element until the shortest length of two numbers. And then the rest of the element add with zero for the rest of the length

(bigger size – smallest size). **We also need to consider the carry if there any. (example 9 + 9 = 18 the carry is 1)**
- Reverse again the result.
- Delete char if there any zero at the front of the string.
- Got the result of two positive numbers.

If the number that wants to be computed is negative then take out the negative sign and decide should you use subtraction(SubstractionIntegersFunction) between two numbers or addition (additionIntegersFunction). If the negative number is smaller than the posistive number then the negative number is unused for the result and vice versa. there will be many variation that will happened. A+B, -A+B, A-B, -A-B = -(A+B).

After getting the result don't forget to add the minus in the front if necessary. If e_total is not NULL then add it in the back of the result with the latter 'e' between it.

SUBTRACTION

If two numbers are positive and integers we can do the subtraction like this **(Remove the dot if there any)**.

SubstractionIntegersFunction

- Reverse both numbers
- Do a subtraction of two numbers element by element until the shortest length of two numbers. And then the rest of the index substract with zero for the rest of the length (bigger size – smallest size). **We also need to consider the carry if there any. (example 0 - 8 = 2 the carry is 1)**
- Reverse again the result.
- Delete char if there any zero the front of the string.
- Got the result of two positive numbers.

If the number that wants to be computed is negative then take out the negative sign and decide you should use subtraction between two numbers(SubstractionIntegersFunction) or addition (additionIntegersFunction). If the negative number is at the second number then the calculation become an addition(A-(-B) = A+B). There will be many variation that will happened. A-B,-A-B = -(A+B), A-(-B) = A+B, -A-(-B) = -A+B.

After getting the result don't forget to add the minus in the front if necessary. If e_total is not NULL then add it in the back of the result with the latter 'e' between it.

MULTIPLICATION

For multiplication is not really complicated. First we have to decide will the result become positive or negative by taking a look at two numbers will it cause negative sign or not. If only one number is negative then the result will be negative. After that we remove the the minus sign and do the multiplication like this **(Remove the dot if there any)**.

MultiplycationIntegersFunction

- Reverse both numbers
- Make another array for the result and fill the array with 0 (zero).
- Using double loop to get all elements from two numbers.
- The calculation we use basically just like normal calculation that we usually did in the paper. You will more understand when look at the code in the part 2.
- And then we got the result of two positive integers.

After getting the result don't forget to add the minus in the front if necessary. If there any number after 'e' then we can just add those two numbers and got the result of the number after 'e' (By default number after e is 0). Then add it in the back of the result with the latter 'e' between it.

DIVISION

For division it's really slow and maybe take longer time if the number is really large. In this project I make the precision maximum is 7. So first we have to decide the result is a negative number or not. If only one number is negative then the result will be negative otherwise not. Then we take out the negative sign **(Remove the dot if there any)**.

DivisionIntegersFunction

- If the denominator is 0 (zero) then it will be error "A number cannot be divided by zero".
- I created a struct called Division to store the result of quotient and the remainder.
- To find the quotient, I used two loops and check one by one if **i** (0,1,2,…..) multiply by the denominator is less than or equal to the numerator and **i+1** is bigger than the numerator. If satisfied then stop.
- The remainder is the result of numerator substract with **i multiply by the denominator.**
- If the remainder is not zero then the result will be decimal.
- I do the loop for the floating point until remainder is 0 (zero) or after looping 7 times.

After getting the result don't forget to add the minus in the front if necessary. If there any number after 'e' then we can just substract those two numbers **(number after 'e' from second number - number after 'e' from first number)** and got the result of the number after 'e' (By default number after e is 0). Then add it in the back of the result with the latter 'e' between it.

Part 2 – Code

For the code I make three c file and two header. So the code is not stacking in one file. And also this is an encapsulation even tho may be not that perfect.

Validation.c

Calculator.c

Mathematic.c

Mathematic.h

Validation.h

Here are the functions that helps the calculator

```c
#pragma once
#include <stdbool.h>
bool isANumber(char *num);
bool isInt(char *num);
bool isDecimal(char *num);
bool isScientific(char *num);
char *deleteUnimportantChar(char *num);
char *insertDotAtIndex(char *num, int index);
char *addingMoreTrailingZero(char *before, int howMany);
char *delete_char(char *str, int k);
```

The function above is provided in the c file. To avoid many pages in this report this function is not shown here but you can find it in the program file.

Below here is the division struct type

```c
typedef struct division
{
    char *quotient;
    char *remainder;
} division;
```

Calculator.c

Include the library

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "validation.h"
#include "mathematic.h"
```

Receive inputs from command line and check if the arguments is contain two numbers and one operator.

```c
if (argc > 4)
{
    printf("The input cannot be interpret as numbers!\n");
    return 0;
}

char *a = (char *)argv[1];
char *op = (char *)argv[2];
char *b = (char *)argv[3];
```

Initialize all variable

```c
int loc_of_e_a = -1;
int loc_of_e_b = -1;

int loc_of_dot_a = -1;
int loc_of_dot_b = -1;

char *eTotal = NULL;

char *e_of_a = "0";
char *e_of_b = "0";
```

Check is the input is valid or not Remove unimportant prefix 0 (zero)

```
if (!isANumber(a) || !(strcmp(op, "*") || strcmp(op, "+") || strcmp(op, "-") || strcmp(op, "/")) || !isANumber(b))
{
    printf("The input cannot be interpret as numbers!\n");
    return 0;
}


a = deleteUnimportantChar(a);
b = deleteUnimportantChar(b);
```

Check the number is integer, decimal or scientific. For decimal I located the dot position. And for scientific I separate the number between ‘e’. The code below is to check for number A but B is also have the same code like this.

```
if (isInt(a))
    {
        ;
    }
    else if (isDecimal(a))
    {
        for (int i = 0; i < strlen(a); i++)
        {
            if (a[i] == '.')
            {
                loc_of_dot_a = i;
                break;
            }
        }
    }
    else if (isScientific(a))
    {
        char *prefixDigitforA;
        for (int i = 0; i < strlen(a); i++)
        {
            if (a[i] == 'e')
            {
                loc_of_e_a = i;
                break;
            }
        }
        // e_of_a = (char *)malloc(sizeof(char) * (strlen(a) - loc_of_e_a - 1));
        e_of_a = a + loc_of_e_a + 1;
        prefixDigitforA = (char *)malloc(sizeof(char *) * (loc_of_e_a));
        snprintf(prefixDigitforA, loc_of_e_a + 1, "%s", a);

        a = prefixDigitforA;
```

```
        if (isInt(a))
        {
            ;
        }
        else
        {
            for (int i = 0; i < strlen(a); i++)
            {
                if (a[i] == '.')
                {
                    loc_of_dot_a = i;
                    break;
                }
            }
            if (loc_of_dot_a > 0)
            {
                // a = delete_char(a, loc_of_dot_a);
            }
        }
    }
```

Then we check if the calculation is for addition or substraction then we need to modify the number after ‘e’ to become equal so it can be calculated. **(if exist at least one scientific number)**

```
if (op == "-" || op == "+")
    {
        if (atoi(e_of_a) > atoi(e_of_b))
        {
            int change = atoi(e_of_a) - atoi(e_of_b);
            eTotal = e_of_b;

            int aaTrailing = 0;
            if (loc_of_dot_a > 0)
            {
                aaTrailing = strlen(a) - loc_of_dot_a - 1;
                a = delete_char(a, loc_of_dot_a);
            }
            a = deleteUnimportantChar(a);
            change -= aaTrailing;

            if (change < 0)
            {

                int offset = strlen(a[0] == '-' ? a + 1 : a) - abs(change);
```

```
                char *res;
                int k = 0;

                if (offset < 0)
                {
                    res = (char *)malloc(sizeof(char) * (abs(offset) + strlen(a)
+ 2));
                    if (a[0] == '-')
                    {
                        res[k++] = '-';
                        a = a + 1;
                    }
                    res[k++] = '0';
                    res[k++] = '.';
                    loc_of_dot_a = k - 1;
                    for (int i = 0; i < abs(offset); i++, k++)
                    {
                        res[k] = '0';
                    }
                    for (int i = 0; i < strlen(a); i++, k++)
                    {
                        res[k] = a[i];
                    }
                    a = res;
                }
                else if (offset == 0)
                {
                    res = (char *)malloc(sizeof(char) * (strlen(a) + 3));
                    if (a[0] == '-')
                    {
                        res[k++] = '-';
                        a = a + 1;
                    }
                    res[k++] = '0';
                    res[k++] = '.';
                    loc_of_dot_a = k - 1;
                    for (int i = 0; i < strlen(a); i++, k++)
                    {
                        res[k] = a[i];
                    }
                    a = res;
                }
                else
                {
                    if (a[0] == '-')
```

```
                    {
                        loc_of_dot_a = offset + 1;
                        a = insertDotAtIndex(a, loc_of_dot_a);
                    }
                    else
                    {
                        loc_of_dot_a = offset;
                        a = insertDotAtIndex(a, loc_of_dot_a);
                    }
                }
            }
            else
            {
                loc_of_dot_a = -1;
                a = addingMoreTrailingZero(a, change);
            }
        }
        else if (atoi(e_of_a) < atoi(e_of_b))
        {
            int change = atoi(e_of_b) - atoi(e_of_a);
            eTotal = e_of_a;

            int bbTrailing = 0;
            if (loc_of_dot_b > 0)
            {
                bbTrailing = strlen(b) - loc_of_dot_b - 1;
                b = delete_char(b, loc_of_dot_b);
            }
            b = deleteUnimportantChar(b);
            change -= bbTrailing;

            if (change < 0)
            {

                int offset = strlen(b[0] == '-' ? b + 1 : b) - abs(change);
                char *res;
                int k = 0;

                if (offset < 0)
                {
                    res = (char *)malloc(sizeof(char) * (abs(offset) + strlen(b)
+ 2));
                    if (b[0] == '-')
                    {
                        res[k++] = '-';
```

```
                b = b + 1;
            }
            res[k++] = '0';
            res[k++] = '.';
            loc_of_dot_b = k - 1;
            for (int i = 0; i < abs(offset); i++, k++)
            {
                res[k] = '0';
            }
            for (int i = 0; i < strlen(b); i++, k++)
            {
                res[k] = b[i];
            }
            b = res;
        }
        else if (offset == 0)
        {
            res = (char *)malloc(sizeof(char) * (strlen(b) + 3));
            if (b[0] == '-')
            {
                res[k++] = '-';
                b = b + 1;
            }
            res[k++] = '0';
            res[k++] = '.';
            loc_of_dot_b = k - 1;
            for (int i = 0; i < strlen(b); i++, k++)
            {
                res[k] = b[i];
            }
            b = res;
        }
        else
        {
            if (b[0] == '-')
            {
                loc_of_dot_b = offset + 1;
                b = insertDotAtIndex(b, loc_of_dot_b);
            }
            else
            {
                loc_of_dot_b = offset;
                b = insertDotAtIndex(b, loc_of_dot_b);
            }
        }
```

```
            }
            else
            {
                loc_of_dot_b = -1;
                b = addingMoreTrailingZero(b, change);
            }
        }
        else
        {
            eTotal = e_of_a;
        }
    }
```

Now if the calculation is addition. Before we do the addition if the number is decimal we need to first take out the decimal and after the calculation done we count again where should we put the dot (‘.’). If the e_total is not null then we also need to add it at the back of the number and put ‘e’ symbol between it.

```
char *answer;
        // if (loc_of_dot_a > 0 || loc_of_dot_b > 0)
        // {
        int aTrailing = 0;
        int bTrailing = 0;
        if (loc_of_dot_a > 0)
        {
            aTrailing = strlen(a) - loc_of_dot_a - 1;
            a = delete_char(a, loc_of_dot_a);
        }
        if (loc_of_dot_b > 0)
        {
            bTrailing = strlen(b) - loc_of_dot_b - 1;
            b = delete_char(b, loc_of_dot_b);
        }
        if (bTrailing > aTrailing)
        {
            a = addingMoreTrailingZero(a, bTrailing - aTrailing);
        }
        if (aTrailing > bTrailing)
        {
            b = addingMoreTrailingZero(b, aTrailing - bTrailing);
        }

        answer = addition(a, b);

        if (bTrailing > aTrailing)
```

```
        {
            answer = insertDotAtIndex(answer, strlen(answer) - bTrailing);
        }
        else if (aTrailing > bTrailing)
        {
            answer = insertDotAtIndex(answer, strlen(answer) - aTrailing);
        }
        else
        {
            if (aTrailing > 0)
            {
                answer = insertDotAtIndex(answer, strlen(answer) - aTrailing);
            }
        }

        if (eTotal == NULL)
        {
            printf("%s\n", answer);
            break;
        }
        printf("%se%s\n", answer, eTotal);
```

While doing the addition we need to consider different variation that could be happen A+B, -A+B, A-B, -A-B = -(A+B). We also need substraction function to complete the calculation. And by doing this we can also know do we need to add negative sign or not.

```
char *addition(char *a, char *b)
{
    if (a[0] == '-' && b[0] != '-') // -a+b
    {
        if (compare(a + 1, b) == 1)
        {
            char *result = subInteger(a + 1, b);
            char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
            res_new[0] = '-';
            for (int i = 0; i < strlen(result); i++)
            {
                res_new[i + 1] = result[i];
            }
            res_new[strlen(result) + 1] = '\0';
            return res_new;
        }
        else
        {
            return subInteger(b, a + 1);
```

```
        }

    }
    if (b[0] == '-' && a[0] != '-') // a+(-b) = a-b
    {
        if (compare(a, b + 1) == 1)
        {
            return subInteger(a, b + 1);
        }
        else
        {
            char *result = subInteger(b + 1, a);
            char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
            res_new[0] = '-';
            for (int i = 0; i < strlen(result); i++)
            {
                res_new[i + 1] = result[i];
            }
            res_new[strlen(result) + 1] = '\0';
            return res_new;
        }
    }
    if (a[0] == '-' && b[0] == '-') // -a+(-b) = -a-b
    {

        char *result = addInteger(a + 1, b + 1);
        char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
        res_new[0] = '-';
        for (int i = 0; i < strlen(result); i++)
        {
            res_new[i + 1] = result[i];
        }
        res_new[strlen(result) + 1] = '\0';
        return res_new;
    }

    return addInteger(a, b);
}
```

Below this is the function to add two positive integers

```
char *addInteger(char *a, char *b)
{
    char *reva = reverse(a);
    char *revb = reverse(b);
```

```
    char *result = (char *)malloc(sizeof(char) * (strlen(a) > strlen(b) ?
strlen(a) + 2 : strlen(b) + 2));

    int carry = 0;
    if (strlen(a) < strlen(b))
    {
        for (int i = 0; i < strlen(a); i++)
        {
            int res = (reva[i] - '0') + (revb[i] - '0') + carry;
            carry = res / 10;
            res = res % 10;
            result[i] = res + '0';
        }
        for (int i = strlen(a); i < strlen(b); i++)
        {
            int res = (revb[i] - '0') + carry;
            carry = res / 10;
            res = res % 10;
            result[i] = res + '0';
        }
        for (int i = strlen(b); i < strlen(b) + 1; i++)
        {
            result[i] = carry + '0';
        }
    }
    else
    {
        for (int i = 0; i < strlen(b); i++)
        {
            int res = (reva[i] - '0') + (revb[i] - '0') + carry;
            carry = res / 10;
            res = res % 10;
            result[i] = res + '0';
        }
        for (int i = strlen(b); i < strlen(a); i++)
        {
            int res = (reva[i] - '0') + carry;
            carry = res / 10;
            res = res % 10;
            result[i] = res + '0';
        }
        for (int i = strlen(a); i < strlen(a) + 1; i++)
        {
            result[i] = carry + '0';
```

```
        }
    }

    result[strlen(a) > strlen(b) ? strlen(a) + 1 : strlen(b) + 1] = '\0';
    int deleteSuffix = 0;
    char *answer = reverse(result);
    for (int i = 0; i < strlen(answer); i++)
    {
        if (answer[i] != '0' || i == strlen(answer) - 1)
        {
            deleteSuffix = i;
            break;
        }
    }
    answer = answer + deleteSuffix;
    return answer;
};
```

For the substraction I also did the same thing like the addition but substraction have different variation so the variation part will also different. A-B,-A-B = -(A+B), A-(-B) = A+B, -A-(-B) = -A+B.

```
char *substraction(char *a, char *b)
{

    if (strcmp(a, b) == 0)
    {
        return "0";
    }
    if (a[0] == '-' && b[0] != '-') // -a-b
    {
        char *result = addInteger(a + 1, b);
        char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
        res_new[0] = '-';
        for (int i = 0; i < strlen(result); i++)
        {
            res_new[i + 1] = result[i];
        }
        res_new[strlen(result) + 1] = '\0';

        return res_new;
    }
    if (b[0] == '-' && a[0] != '-') // a-(-b) = a+b
    {
        return addInteger(b + 1, a);
```

```
    }
    if (a[0] == '-' && b[0] == '-') // -a-(-b) = -a+b
    {
        if (compare(a + 1, b + 1) == 1)
        {
            char *result = subInteger(a + 1, b + 1);
            char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
            res_new[0] = '-';
            for (int i = 0; i < strlen(result); i++)
            {
                res_new[i + 1] = result[i];
            }
            res_new[strlen(result) + 1] = '\0';
            return res_new;
        }
        else
        {
            return subInteger(b + 1, a + 1);
        }
    }
    if (compare(a, b) == 1)
    {
        return subInteger(a, b);
    }
    else
    {
        char *result = subInteger(b, a);
        char *res_new = (char *)malloc(sizeof(char) * (strlen(result) + 2));
        res_new[0] = '-';
        for (int i = 0; i < strlen(result); i++)
        {
            res_new[i + 1] = result[i];
        }
        res_new[strlen(result) + 1] = '\0';
        return res_new;
    }
}
```

Below this is the function to substract two positive integers

```
char *subInteger(char *a, char *b) // a>b
{

    bool carry = false;
    char *reva = reverse(a);
```

```
    char *revb = reverse(b);

    char *result = (char *)malloc(sizeof(char) * (strlen(a) + 1));

    for (int i = 0; i < strlen(revb); i++)
    {
        int first = reva[i] - '0';
        int second = revb[i] - '0';
        if (carry && first > 0)
        {
            first--;
            carry = false;
        }
        if (carry && first == 0)
        {
            first = 9;
            carry = true;
        }
        if (first < second)
        {
            first += 10;
            carry = true;
        }

        result[i] = (first - second) + '0';
    }

    for (int i = strlen(revb); i < strlen(reva); i++)
    {
        int first = reva[i] - '0';
        if (carry && first > 0)
        {
            first--;
            carry = false;
        }
        result[i] = first + '0';
    }

    result[strlen(a)] = '\0';
    int deleteSuffix = 0;
    char *answer = reverse(result);
    for (int i = 0; i < strlen(answer); i++)
    {
        if (answer[i] != '0' || i == strlen(answer) - 1)
        {
```

```
            deleteSuffix = i;
            break;
        }
    }
    answer = answer + deleteSuffix;
    return answer;
}
```

For the multiplication we first need to take out the dot if there any. And after we do the multiplication we then need to calculate the decimal since we have store the info of the decimal before we compute.

```
char *answer;
        int aTrailing = 0;
        int bTrailing = 0;
        if (loc_of_dot_a > 0)
        {
            aTrailing = strlen(a) - loc_of_dot_a - 1;
            a = delete_char(a, loc_of_dot_a);
        }
        if (loc_of_dot_b > 0)
        {
            bTrailing = strlen(b) - loc_of_dot_b - 1;
            b = delete_char(b, loc_of_dot_b);
        }
        int totalTrailing = aTrailing + bTrailing;
        answer = multiplication(a, b);
```

In doing multiplication we need to determine the result is negative or not. If there only one number negative then the result will be negative.

```
char *multiplication(char *a, char *b)
{
    bool sign = false;
    char *answer;
    if (a[0] == '-' && b[0] == '-')
    {
        answer = multiplyInt(a + 1, b + 1);
    }
    else if (a[0] == '-')
    {
        sign = true;
        answer = multiplyInt(a + 1, b);
    }
```

```
    else if (b[0] == '-')
    {
        sign = true;
        answer = multiplyInt(a, b + 1);
    }
    else
    {
        answer = multiplyInt(a, b);
    }

    if (sign)
    {
        char *res_new = (char *)malloc(sizeof(char) * (strlen(answer) + 2));
        res_new[0] = '-';
        for (int i = 0; i < strlen(answer); i++)
        {
            res_new[i + 1] = answer[i];
        }
        res_new[strlen(answer) + 1] = '\0';

        return res_new;
    }
    return answer;
}
```

And then below is the code to multiply two positive integers.

```
char *multiplyInt(char *a, char *b)
{

    char *reva = reverse(a);
    char *revb = reverse(b);

    int *revres = (int *)calloc(strlen(a) + strlen(b), sizeof(int));
    char *revresChar = (char *)malloc((strlen(a) + strlen(b) + 1) *
sizeof(char));

    for (int i = 0; i < strlen(a); i++)
    {
        for (int j = 0; j < strlen(b); j++)
        {

            revres[i + j] = revres[i + j] + (reva[i] - '0') * (revb[j] - '0');
        }
    }
```

```
    for (int i = 0; i < strlen(a) + strlen(b); i++)
    {
        if (revres[i] > 9)
        {
            revres[i + 1] = revres[i + 1] + revres[i] / 10;
            revres[i] = revres[i] % 10;
        }
    }
    for (int i = strlen(a) + strlen(b) - 1; i >= 0; i--)
    {
        revresChar[i] = revres[i] + '0';
    }
    revresChar[strlen(a) + strlen(b)] = '\0';
    free(revres);
    int deleteSuffix = 0;
    char *answer = reverse(revresChar);
    for (int i = 0; i < strlen(answer); i++)
    {
        if (answer[i] != '0' || i == strlen(answer) - 1)
        {
            deleteSuffix = i;
            break;
        }
    }
    answer = answer + deleteSuffix;

    return answer;
}
```

After we got the result then we need to calculate where the decimal is and then decide if it need number after ‘e’.

```
if (totalTrailing == 0)
        {
            if (strcmp(addition(e_of_a, e_of_b), "0") == 0)
            {
                printf("%s\n", answer);
            }
            else
            {
                printf("%se%s\n", answer, addition(e_of_a, e_of_b));
            }

            break;
```

```
        }

        int offset = strlen(answer[0] == '-' ? answer + 1 : answer) -
totalTrailing;
        char *res;
        int k = 0;

        if (offset < 0)
        {
            res = (char *)malloc(sizeof(char) * (abs(offset) + strlen(answer) +
2));
            if (answer[0] == '-')
            {
                res[k++] = '-';
                answer = answer + 1;
            }
            res[k++] = '0';
            res[k++] = '.';

            for (int i = 0; i < abs(offset); i++, k++)
            {
                res[k] = '0';
            }
        }
        else if (offset == 0)
        {
            res = (char *)malloc(sizeof(char) * (strlen(answer) + 3));
            if (answer[0] == '-')
            {
                res[k++] = '-';
                answer = answer + 1;
            }
            res[k++] = '0';
            res[k++] = '.';
        }
        else
        {
            if (answer[0] == '-')
            {
                if (addition(e_of_a, e_of_b) == "0")
                {
                    char *help = (char *)malloc(sizeof(char) * (offset + 1));
                    snprintf(help, offset + 2, "%s", insertDotAtIndex(answer,
offset));
                    answer = answer + offset;
```

```
                    printf("-%s%s\n", help,
reverse(deleteUnimportantChar(reverse(answer))));
                }
                else
                {
                    char *help = (char *)malloc(sizeof(char) * (offset + 1));
                    snprintf(help, offset + 2, "%s", insertDotAtIndex(answer,
offset));
                    answer = answer + offset;
                    printf("-%s%se%s\n", help,
reverse(deleteUnimportantChar(reverse(answer))), addition(e_of_a, e_of_b));
                }
            }
            else
            {
                if (strcmp(addition(e_of_a, e_of_b), "0") == 0)
                {
                    char *help = (char *)malloc(sizeof(char) * (offset + 1));
                    snprintf(help, offset + 2, "%s", insertDotAtIndex(answer,
offset));
                    answer = answer + offset;
                    printf("%s%s\n", help,
reverse(deleteUnimportantChar(reverse(answer))));
                }
                else
                {
                    char *help = (char *)malloc(sizeof(char) * (offset + 1));
                    snprintf(help, offset + 2, "%s", insertDotAtIndex(answer,
offset));
                    answer = answer + offset;
                    printf("%s%se%s\n", help,
reverse(deleteUnimportantChar(reverse(answer))), addition(e_of_a, e_of_b));
                }
            }

            break;
        }
        for (int i = 0; i < strlen(answer); i++, k++)
        {
            res[k] = answer[i];
        }
        res[k] = '\0';
        if (strcmp(addition(e_of_a, e_of_b), "0") == 0)
        {
            printf("%s\n", res);
```

```
        }
        else
        {
            printf("%se%s\n", res, addition(e_of_a, e_of_b));
        }

        break;
```

For the division it's not diferent with multiply in the first process and also to determine the result will be negative or not. But before we do division we need to check if denominator is not 0 (zero).

```
    case '/':
    {
        if (strcmp(substraction(b, "0"), "0")==0)
        {
            printf("A number cannot be divided by zero.\n");
            break;
        }
```

And then below is the code for the division part

```
Division oneTimeDiv(char *a, char *b) // (a/b)
{
    Division result;
    result.quotient = "0";
    result.remainder = "";

    if (subInteger(b, "0") == "0")
    {
        result.quotient = "wrong";
    }
    else if (subInteger(a, "0") == "0")
    {
        result.quotient = "0";
    }
    else
    {
        for (char *i = "0";; i = addInteger(i, "1"))
        {
            if ((compare(multiplyInt(b, i), a) == -1 || compare(multiplyInt(b,
i), a) == 0) && (compare(multiplyInt(b, addInteger(i, "1")), a) == 1))
            {
                result.quotient = i;
```

```
                result.remainder = subInteger(a, multiplyInt(b, i));
                break;
            }
        }
    }

    return result;
}
```

And after we got the result we need to compute the decimal location and the number after ‘e’ if there any.

For further detail, the original code is provided in the attached file.

Part 3 – Result & Verification

How to run the program?

```
P-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/testok/sustechCppC
1$ gcc -c calculator.c
P-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/testok/sustechCppC
1$ gcc -c validation.c
P-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/testok/sustechCppC
1$ gcc -c mathematic.c
P-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/testok/sustechCppC
1$ gcc calculator.o validation.o mathematic.o -o calculator
P-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/testok/sustechCppC
```

Test case #1

2 + 3 = 5

2 – 3 = -1

2 * 3 = 6

2 / 3 = 0.6666666

```
nopyesok@DESKTOP-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/
testok/sustechCppClass/cs205/project1$ ./calculator 2 + 3
5
nopyesok@DESKTOP-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/
testok/sustechCppClass/cs205/project1$ ./calculator 2 - 3
-1
nopyesok@DESKTOP-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/
testok/sustechCppClass/cs205/project1$ ./calculator 2 '*' 3

6
nopyesok@DESKTOP-TEL06N0:/mnt/c/Users/Livia/Documents/bugr/
testok/sustechCppClass/cs205/project1$ ./calculator 2 / 3
0.6666666
```

Test case #2

3.14 / 0

A number cannot be divided by zero.

```
k/sustechCppClass/cs205/project1$ ./calculator 3.14 / 0
A number cannot be divided by zero.
```

Test case #3

a '*' 2

The input cannot be interpret as numbers!

```
/cs205/project1$ .
/calculator a '*' 2
The input cannot be interpret as numbers!
```

Test case #4

987654321 * 987654321

975461057789971041

```
/cs205/project1$ ./calculator 987654321 '*' 987654321
975461057789971041
```

Test case #5

1.0e200 * 1.0e200

1.0e400

```
/cs205/project1$ ./calculator 1.0e200 '*' 1.0e200
1.0e400
```

Part 4 – Difficulties & Solutions

a. I don't know how to initialize the size of the input, so I just relize that the input is from command line so I don't need to worry about the size.
b. The input for multiply is asterix and some how the computer cannot detect asterix *, so I have to add single quotation '*'.
c. The memory management is really troublesome. If the length we provided for result or the new changes not right then it will also impact other value since the memory size is not correct. So I solve this by provided size that at least correct or maybe more.

d. I malloc the memory size and the result digit after the actual digit number is strange, so I solved this by adding ‘\0’ after the last digit so it’s stop right there (null terminator).
e. The big number calculation will exceed the limit of integer type or even long long type so the calculation was made by an array.