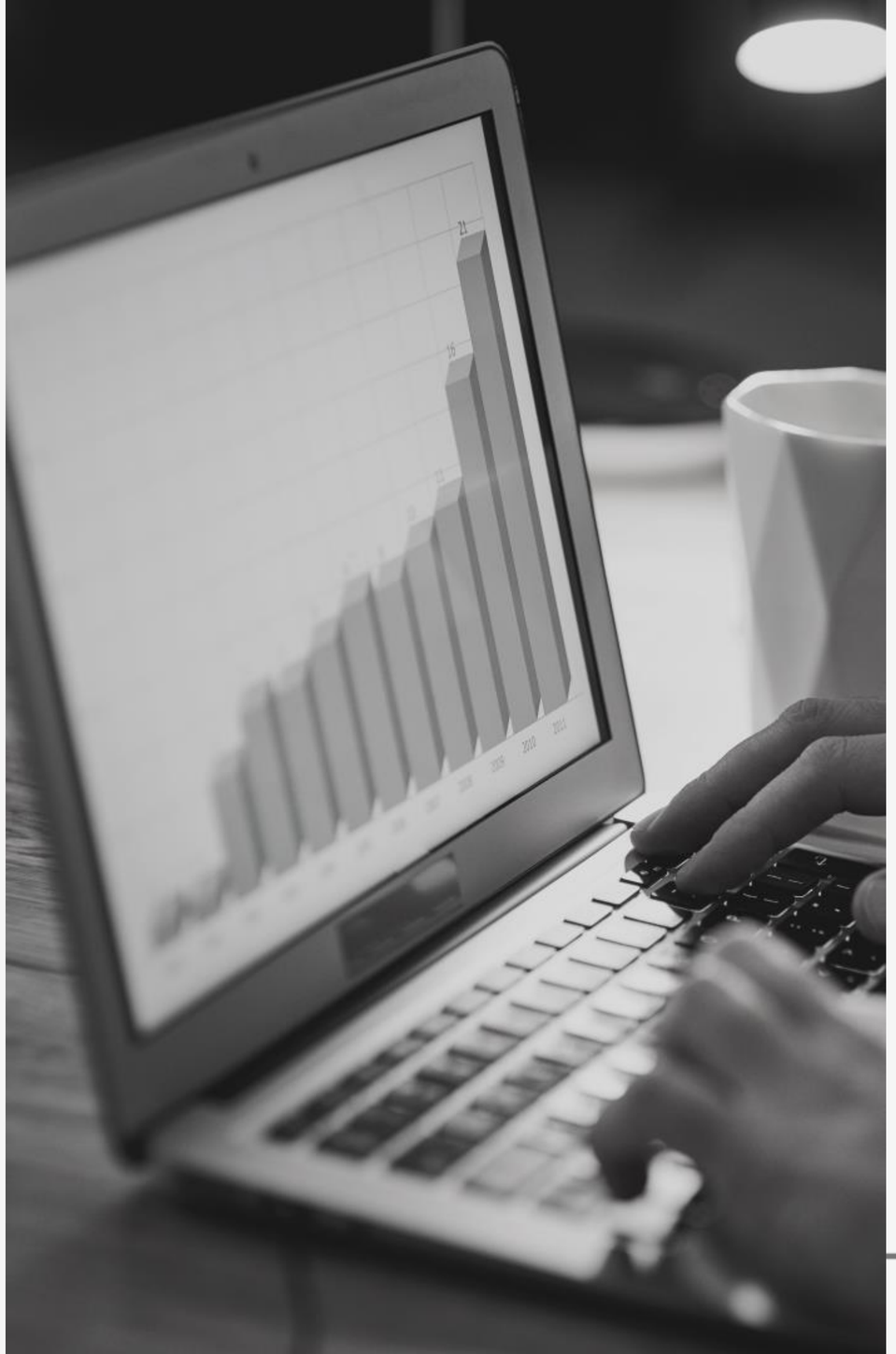
20102122 Jeong Hyoan
20102123 Choi JinA
20102124 Chu Yulim

Computer Systems

Project 1



Contents

01

Number Representation

The maximum range of double types

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <float.h>
#include <string.h>
#define SIZE 100

int show_maximumDouble() {
    double doubleMax = DBL_MAX;
    printf("The largest double value : %f\n\n", doubleMax);
    return 0;
}

int howToStoreinDouble() {
    double exDouble = 12121212121212121212121212.1212121212121212;
    double* address = &exDouble;

    printf("The way how to represent(double) : %f \nThe address : %p\n\n", exDouble, address);

    return 0;
}
```

```
> Executing task: cmd /C 'd:\Computer Systems\C언어\represent' <

The largest double value : 1797693134862315700000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000.000000

The way how to represent(double) : 121212121212121220000000.000000
The address : 0061FF00
```

01 Check the maximum value of the number that can be expressed through the double type with 'DBL_MAX'

02 Check how the number exceeding the double-type maximum range is stored and printed

“

How can we represent a number that *exceeds the maximum range* of the data type?

”

Representing numbers by the array

```c
int separate() {
    char num1[] = "12121212121212121212121212.1212121212121212";
    char* sepNum1[2] = { NULL, };

    char* token1 = strtok(num1, ".");

    sepNum1[0] = token1;
    token1 = strtok(NULL, ".");
    sepNum1[1] = token1;

    printf("%s.", sepNum1[0]);
    printf("%s\n", sepNum1[1]);
```

01

Make a number into a string

- Make the number into a num1 string.
- Based on '.', num1 is divided into an integer part and a fractional part.

```c
//num1 정수부
int i = 0, j = 1;
int num1L[SIZE];

while (sepNum1[0][i] != NULL) {
    i++;
}

for (j; j<= i; j++) {
    num1L[i-j] = sepNum1[0][j-1] - 48;
}

if (i != SIZE) {
    for (i; i <= SIZE-1; i++) {
        num1L[i] = 0;
    }
}
```

02

Convert the integer part of num1 to an int array(num1L)

- Change char to int by subtracting 48('0')
- Fill the rest with zero to match the number of digits (index)

```c
//num1 소수부
i = 0, j = 1;
int num1R[SIZE];

while (sepNum1[1][i] != NULL) {
    i++;
}
for (j; j <= i; j++) {
    num1R[SIZE - j] = sepNum1[1][j - 1] - 48;
}
i = 0;
if (j != SIZE) {
    for (i; i <= SIZE -j; i++) {
        num1R[i] = 0;
    }
}
```

03

Convert the fractional part of num1 to an int array(num1R)

- Change the fractional part, such as the integer part in step 2.
- Another number, num2, is also converted like num1

02

Addition & Subtraction

Addition implementation

- Represent the two numbers as strings
- Modify the string according to the sign
- Convert string to int array
  (Match the number of digits through 0)

```c
#define SIZE 100

int main() {
    int num1zeroLength = SIZE - 1;
    int num2zeroLength = SIZE - 1;
    int subCompare;

    char num1[SIZE];
    char num2[SIZE];

    printf("Please enter the floating number : ");
    scanf(" %s", num1);
    printf("Please enter the floating number : ");
    scanf(" %s", num2);

    int num1Sign = 1;
    int num2Sign = 1;
    int resSign = 1;

    if (num1[0] == '-') {
        for (int i = 1; i < strlen(num1); i++) {
            num1[i - 1] = num1[i];
        }
        num1[strlen(num1) - 1] = NULL;
        num1Sign *= -1;
    }
    if (num2[0] == '-') {
        for (int i = 1; i < strlen(num2); i++) {
            num2[i - 1] = num2[i];
        }
        num2Sign *= -1;
        num2[strlen(num2) - 1] = NULL;
    }
```

```c
//num1 정수부
int i = 0, j = 1;
int num1L[SIZE];

while (sepNum1[0][i] != NULL) {
    i++;
}

for (j; j <= i; j++) {
    num1L[i - j] = sepNum1[0][j - 1] - 48;
}

if (i != SIZE) {
    for (i; i <= SIZE - 1; i++) {
        num1L[i] = 0;
    }
}

//num1 소수부
i = 0, j = 1;
int num1R[SIZE];

while (sepNum1[1][i] != NULL) {
    i++;
}
for (j; j <= i; j++) {
    num1R[SIZE - j] = sepNum1[1][j - 1] - 48;
}
i = 0;
if (j != SIZE) {
    for (i; i <= SIZE - j; i++) {
        num1R[i] = 0;
    }
}
```

Addition implementation

If two numbers have the same sign,

```
//sum
int sumResR[SIZE];
int sumResL[SIZE];

if (num1Sign > 0 && num2Sign > 0) { // num1과 num2가 모두 양수일때
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResR[n] = num1R[n] + num2R[n];
        if (n != SIZE - 1 && sumResR[n] >= 10) {
            num1R[n + 1] += 1;
            sumResR[n] -= 10;
        }
        else if (n == SIZE - 1 && sumResR[n] >= 10) {
            num1L[0] += 1;
            sumResR[n] -= 10;
        }
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResL[n] = num1L[n] + num2L[n];
        if (n != SIZE - 1 && sumResL[n] >= 10) {
            num1L[n + 1] += 1;
            sumResL[n] -= 10;
        }
    }
    resSign = 1;
}
```

When both are positive

```
else if (num1Sign < 0 && num2Sign < 0) { // num1과 num2가 모두 음수일때
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResR[n] = num1R[n] + num2R[n];
        if (n != SIZE - 1 && sumResR[n] >= 10) {
            num1R[n + 1] += 1;
            sumResR[n] -= 10;
        }
        else if (n == SIZE - 1 && sumResR[n] >= 10) {
            num1L[0] += 1;
            sumResR[n] -= 10;
        }
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResL[n] = num1L[n] + num2L[n];
        if (n != SIZE - 1 && sumResL[n] >= 10) {
            num1L[n + 1] += 1;
            sumResL[n] -= 10;
        }
    }
    resSign = -1;
}
```

When both are negative

Addition implementation

If two numbers have different signs,

```
else if (num1Sign > 0 && num2Sign < 0) { // num2가 음수일때
    while (num1L[num1zeroLength] == 0) { num1zeroLength--; }
    while (num2L[num2zeroLength] == 0) { num2zeroLength--; }

    if (num1zeroLength > num2zeroLength) {
            resSign = 1;
        }
    else if (num1zeroLength == num2zeroLength) {
        subCompare = num1zeroLength;
        while (num1L[subCompare] >= num2L[subCompare]) {

            subCompare--;
        }
        if (subCompare < num1zeroLength) { resSign = 1; }
        else { resSign = -1; }
    }
    else { resSign = -1; }
```

Determining the sign of the result value

```
if (resSign == 1) {
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResR[n] = num1R[n] - num2R[n];
        if (n != SIZE - 1 && sumResR[n] < 0) {
            num1R[n + 1] -= 1;
            sumResR[n] += 10;
        }
        else if (n == SIZE - 1 && sumResR[n] < 0) {
            num1L[0] -= 1;
            sumResR[n] += 10;
        }
    }
    if (num1L[0] == -1) {
        num1L[1] -= 1;
        num1L[0] += 10;
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResL[n] = num1L[n] - num2L[n];
        if (n != SIZE - 1 && sumResL[n] < 0) {
            num1L[n + 1] -= 1;
            sumResL[n] += 10;
        }
    }
}
```

When the result is positive

```
else {
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResR[n] = num2R[n] - num1R[n];
        if (n != SIZE - 1 && sumResR[n] < 0) {
            num2R[n + 1] -= 1;
            sumResR[n] += 10;
        }
        else if (n == SIZE - 1 && sumResR[n] < 0) {
            num2L[0] -= 1;
            sumResR[n] += 10;
        }
    }
    if (num2L[0] == -1) {
        num2L[1] -= 1;
        num2L[0] += 10;
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        sumResL[n] = num2L[n] - num1L[n];
        if (n != SIZE - 1 && sumResL[n] < 0) {
            num2L[n + 1] -= 1;
            sumResL[n] += 10;
        }
    }
}
```

When the result is negative

11

Addition implementation

Output the result

```
int zeroLength = SIZE - 1;

// 더하기 결과 출력
while (sumResL[zeroLength] == 0) { zeroLength--; }
printf("Result (sum) : ");
if (resSign < 0) {
    printf("-");
}
for (int n = zeroLength; n >= 0; n--) { printf("%d", sumResL[n]); }
printf(".");

zeroLength = 0;
while (sumResR[zeroLength] == 0) { zeroLength++; }
for (int n = SIZE - 1; n >= zeroLength; n--) { printf("%d", sumResR[n]); }
printf("0\n");
```

Minus sign output & Unnecessary zero removal

```
> Executing task: cmd /C 'd:\Computer Systems\C언어\addition' <

Please enter the floating number : 12121212121212121212121212.1212121212121212
Please enter the floating number : 98989898989898989898989898.989898989898989898989898
Result (sum) : 991111111111111111111111111.111111111111110898980

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

> Executing task: cmd /C 'd:\Computer Systems\C언어\addition' <

Please enter the floating number : 11111111111111111.1111111111111
Please enter the floating number : -77777777777777777.77777777777777
Result (sum) : -666666666666666666.66666666666660

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

The result of calculating with following numbers and random numbers

Subtraction implementation

- Represent the two numbers as strings
- Modify the string according to the sign
- Convert string to int array
  (Match the number of digits through 0)

```c
int main() {
    int num1zeroLength = SIZE - 1;
    int num2zeroLength = SIZE - 1;
    int subCompare;

    char num1[SIZE];
    char num2[SIZE];

    printf("Please enter the floating number : ");
    scanf(" %s", num1);
    printf("Please enter the floating number : ");
    scanf(" %s", num2);

    int num1Sign = 1;
    int num2Sign = 1;
    int resSign = 1;

    if (num1[0] == '-') {
        for (int i = 1; i < strlen(num1); i++) {
            num1[i - 1] = num1[i];
        }
        num1[strlen(num1) - 1] = NULL;
        num1Sign *= -1;
    }
    if (num2[0] == '-') {
        for (int i = 1; i < strlen(num2); i++) {
            num2[i - 1] = num2[i];
        }
        num2Sign *= -1;
        num2[strlen(num2) - 1] = NULL;
    }
```

```c
char* sepNum1[2] = { NULL, };
char* token1 = strtok(num1, ".");

sepNum1[0] = token1;
token1 = strtok(NULL, ".");
sepNum1[1] = token1;

char* sepNum2[2] = { NULL, };
char* token2 = strtok(num2, ".");

sepNum2[0] = token2;
token2 = strtok(NULL, ".");
sepNum2[1] = token2;

//num1 정수부
int i = 0, j = 1;
int num1L[SIZE];

while (sepNum1[0][i] != NULL) {
    i++;
}

for (j; j <= i; j++) {
    num1L[i - j] = sepNum1[0][j - 1] - 48;
}

if (i != SIZE) {
    for (i; i <= SIZE - 1; i++) {
        num1L[i] = 0;
    }
}
```

Subtraction implementation

If two numbers have the same sign,

```
//sub
int subResR[SIZE];
int subResL[SIZE];
if (num1Sign < 0 && num2Sign < 0) { // num1, num2 모두 음수일때
    while (num1L[num1zeroLength] == 0) { num1zeroLength--; }
    while (num2L[num2zeroLength] == 0) { num2zeroLength--; }

    if (num2zeroLength > num1zeroLength) {
        resSign = 1;
    }
    else if (num2zeroLength == num1zeroLength) {
        subCompare = num2zeroLength;
        while (num2L[subCompare] >= num1L[subCompare]) {

            subCompare--;
        }
        if (subCompare < num2zeroLength) { resSign = 1; }
        else { resSign = -1; }
    }
    else { resSign = -1; }
```

Determining the sign of the result value

```
if (resSign == 1) {
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num2R[n] - num1R[n];
        if (n != SIZE - 1 && subResR[n] < 0) {
            num2R[n + 1] -= 1;
            subResR[n] += 10;
        }
        else if (n == SIZE - 1 && subResR[n] < 0) {
            num2L[0] -= 1;
            subResR[n] += 10;
        }
    }
    if (num2L[0] == -1) {
        num2L[1] -= 1;
        num2L[0] += 10;
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num2L[n] - num1L[n];
        if (n != SIZE - 1 && subResL[n] < 0) {
            num2L[n + 1] -= 1;
            subResL[n] += 10;
        }
    }
}
```

When the result is positive

```
else {
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num1R[n] - num2R[n];
        if (n != SIZE - 1 && subResR[n] < 0) {
            num1R[n + 1] -= 1;
            subResR[n] += 10;
        }

        else if (n == SIZE - 1 && subResR[n] < 0) {
            num1L[0] -= 1;
            subResR[n] += 10;
        }
    }
    if (num1L[0] == -1) {
        num1L[1] -= 1;
        num1L[0] += 10;
    }

    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num1L[n] - num2L[n];
        if (n != SIZE - 1 && subResL[n] < 0) {
            num1L[n + 1] -= 1;
            subResL[n] += 10;
        }
    }
}
```

When the result is negative

Subtraction implementation

If two numbers have the same sign,

```
else { // num1, num2 모두 양수일때
    while (num1L[num1zeroLength] == 0) { num1zeroLength--; }
    while (num2L[num2zeroLength] == 0) { num2zeroLength--; }

    if (num1zeroLength > num2zeroLength) {
        resSign = 1;
    }
    else if (num1zeroLength == num2zeroLength) {
        subCompare = num1zeroLength;
        while (num1L[subCompare] >= num2L[subCompare]) {

            subCompare--;
        }
        if (subCompare < num1zeroLength) { resSign = 1; }
        else { resSign = -1; }
    }
    else { resSign = -1; }
```

Determining the sign of the result value

```
if (resSign == 1) {
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num1R[n] - num2R[n];
        if (n != SIZE - 1 && subResR[n] < 0) {
            num1R[n + 1] -= 1;
            subResR[n] += 10;
        }
        else if (n == SIZE - 1 && subResR[n] < 0) {
            num1L[0] -= 1;
            subResR[n] += 10;
        }
    }
    if (num1L[0] == -1) {
        num1L[1] -= 1;
        num1L[0] += 10;
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num1L[n] - num2L[n];
        if (n != SIZE - 1 && subResL[n] < 0) {
            num1L[n + 1] -= 1;
            subResL[n] += 10;
        }
    }
}
```

When the result is positive

```
else {
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num2R[n] - num1R[n];
        if (n != SIZE - 1 && subResR[n] < 0) {
            num2R[n + 1] -= 1;
            subResR[n] += 10;
        }
        if (num2L[0] == -1) {
            num2L[1] -= 1;
            num2L[0] += 10;
        }
        else if (n == SIZE - 1 && subResR[n] < 0) {
            num2L[0] -= 1;
            subResR[n] += 10;
        }
    }

    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num2L[n] - num1L[n];
        if (n != SIZE - 1 && subResL[n] < 0) {
            num2L[n + 1] -= 1;
            subResL[n] += 10;
        }
    }
}
```

When the result is negative

Subtraction implementation

If two numbers have the different signs,

```
else if (num2Sign < 0) { // num2가 음수일때
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num1R[n] + num2R[n];
        if (n != SIZE - 1 && subResR[n] >= 10) {
            num1R[n + 1] += 1;
            subResR[n] -= 10;
        }
        else if (n == SIZE - 1 && subResR[n] >= 10) {
            num1L[0] += 1;
            subResR[n] -= 10;
        }
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num1L[n] + num2L[n];
        if (n != SIZE - 1 && subResL[n] >= 10) {
            num1L[n + 1] += 1;
            subResL[n] -= 10;
        }
    }
    resSign = 1;
}
```

When num2 is negative

```
else if (num1Sign < 0) { // num1이 음수일때
    for (int n = 0; n <= SIZE - 1; n++) {
        subResR[n] = num1R[n] + num2R[n];
        if (n != SIZE - 1 && subResR[n] >= 10) {
            num1R[n + 1] += 1;
            subResR[n] -= 10;
        }
        else if (n == SIZE - 1 && subResR[n] >= 10) {
            num1L[0] += 1;
            subResR[n] -= 10;
        }
    }
    for (int n = 0; n <= SIZE - 1; n++) {
        subResL[n] = num1L[n] + num2L[n];
        if (n != SIZE - 1 && subResL[n] >= 10) {
            num1L[n + 1] += 1;
            subResL[n] -= 10;
        }
    }
    resSign = -1;
}
```

When num1 is negative

Subtraction implementation

Output the result

```c
// 빼기 결과 출력
int zeroLength = SIZE - 1;
printf("Result (sub) : ");

if (resSign < 0) {
    printf("-");
}
while (subResL[zeroLength] == 0) { zeroLength--; }
for (int n = zeroLength; n >= 0; n--) { printf("%d", subResL[n]); }
printf(".");

zeroLength = 0;
while (subResR[zeroLength] == 0) { zeroLength++; }
for (int n = SIZE - 1; n >= zeroLength; n--) { printf("%d", subResR[n]); }
printf("0\n");
```

Minus sign output & Unnecessary zero removal

```
> Executing task: cmd /C 'd:\Computer Systems\C언어\subtraction' <

Please enter the floating number : 12121212121212121212121212.1212121212121212
Please enter the floating number : 98989898989898989898989898.98989898989898989898
Result (sub) : -98868686868686868686868686.868686868686868690898980

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

> Executing task: cmd /C 'd:\Computer Systems\C언어\subtraction' <

Please enter the floating number : 123123123123132123.123123123123123
Please enter the floating number : -111111111111111111.111111111111111
Result (sub) : 234234234234243234.2342342342342340
```

The result of calculating with following numbers and random numbers

03

Multiplication

Multiplication implementation

```
int sign = 1;

if (num1[0] == '-') {
    for (int i = 1; i < strlen(num1); i++) {
        num1[i - 1] = num1[i];
    }
    num1[strlen(num1) - 1] = NULL;
    sign *= -1;
}
if (num2[0] == '-') {
    for (int i = 1; i < strlen(num2); i++) {
        num2[i - 1] = num2[i];
    }
    sign *= -1;
    num2[strlen(num2) - 1] = NULL;
}

int mulRes[SIZE * 2] = { 0 };
```

Determining the sign of the result value

```
for (int i = strlen(num1); i >= 0; i--) {
    if (num1[i] != '.') {
        num1RNum += 1;
    }
    else {
        break;
    }
}

for (int i = strlen(num2); i >= 0; i--) {
    if (num2[i] != '.') {
        num2RNum += 1;
    }
    else {
        break;
    }
}

int resRNum = num1RNum + num2RNum - 2;
```

Finding the number of numbers after the decimal point

Multiplication implementation

```
for (int i = 0; i < strlen(num1) - num1RNum; i++) {
    newNum1[i] = num1[i] - 48;
}
for (int i = strlen(num1)-2; i >= strlen(num1) - num1RNum; i--) {
    newNum1[i] = num1[i+1] - 48;
}

for (int i = 0; i < strlen(num2) - num2RNum; i++) {
    newNum2[i] = num2[i] - 48;
}
for (int i = strlen(num2) - 2; i >= strlen(num2) - num2RNum; i--) {
    newNum2[i] = num2[i + 1] - 48;
}
```

Creating an int array minus decimal point

```
for (int i = strlen(num1) - 2; i >= 0; i--)
{
    for (int j = strlen(num2) - 2; j >= 0; j--)
    {
        mulRes[i + j] += newNum1[i] * newNum2[j];
    }

}

for (int i = strlen(num1) + strlen(num2) - 4; i > 0; i--)
{
    if (mulRes[i] >= 10) {
        mulRes[i - 1] += mulRes[i] / 10;
        mulRes[i] %= 10;
    }
}
```

Calculation by index

Multiplication implementation

Output the result

```
printf("Result (mult) : ");

int existNum = strlen(num1) + strlen(num2) - 4;
if (sign < 0) {
    printf("-");
}
for (int i = 0; i <= existNum - resRNum; i++)
    printf("%d", mulRes[i]);
printf(".");
for (int i = existNum - resRNum + 1; i <= existNum; i++)
    printf("%d", mulRes[i]);

return 0;
```

Minus sign output & Adding a decimal point

```
> Executing task: cmd /C 'd:\Computer Systems\C언어\multiplication' <

Please enter the floating number : 12121212121212121212121212.1212121212121212
Please enter the floating number : 9898989898989898989898989898.98989898989898989898
Result (mult) : 11998775635139271502907866544230180593815757575745.576981940618304254667891033
152739516376
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

```
> Executing task: cmd /C 'd:\Computer Systems\C언어\multiplication' <

Please enter the floating number : 1.5
Please enter the floating number : -2222222222222222.22222222222222222222
Result (mult) : -3333333333333333.333333333333333333333330
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

The result of calculating with following numbers and random numbers

04

Division

: Repeated subtraction

Division implementation

```
int resSign = 1;

if (num1[0] == '-') {
    for (int i = 1; i < strlen(num1); i++) {
        num1[i - 1] = num1[i];
    }
    num1[strlen(num1) - 1] = NULL;
    resSign *= -1;
}
if (num2[0] == '-') {
    for (int i = 1; i < strlen(num2); i++) {
        num2[i - 1] = num2[i];
    }
    resSign *= -1;
    num2[strlen(num2) - 1] = NULL;
}



for (int i = strlen(num1); i >= 0; i--) {
    if (num1[i] != '.') {
        num1RNum += 1;
    }
    else {
        break;
    }
}
```

Determine the sign of the result value

```
void insert(int list[], int cnt, int size)     // in
{
    for (int i = 0; i < cnt; i++) {
        for (int j = size - 1; j >= 0; j--) {
            list[j + 1] = list[j];
            list[j] = 0;
```

```
for (int i = 0; i < strlen(num1) - num1RNum; i++) {
    newNum1[i] = num1[i] - 48;
}
for (int i = strlen(num1) - 1; i > strlen(num1) - num1RNum; i--) {
    newNum1[i - 1] = num1[i] - 48;
}

for (int i = 0; i < strlen(num2) - num2RNum; i++) {
    newNum2[i] = num2[i] - 48;
}
for (int i = strlen(num2) - 1; i > strlen(num2) - num2RNum; i--) {
    newNum2[i - 1] = num2[i] - 48;
}


// 자릿수가 다르면 인덱스 0 에 0 추가
if (strlen(num1) - num1RNum < strlen(num2) - num2RNum) {
    int cnt = (strlen(num2) - num2RNum) - (strlen(num1) - num1RNum);
    insert(newNum1, cnt, strlen(num1)-1);

}
else if (strlen(num1) - num1RNum > strlen(num2) - num2RNum) {
    int cnt = (strlen(num1) - num1RNum) - (strlen(num2) - num2RNum);
    insert(newNum2, cnt, strlen(num1) - 1);

}
```

Make the numbers an int array and place by decimal point.

Division implementation

```
for (int i = 0; i <= SIZE - 1; i++) {
    reNum1[i] = newNum1[SIZE - 1 - i];
}
for (int i = 0; i <= SIZE - 1; i++) {
    reNum2[i] = newNum2[SIZE - 1 - i];
}

// 나누기
int subRes[SIZE] = { 0 };
int temp[SIZE] = { 0 };
int divRes[SIZE] = { NULL };
int tempzeroLength;
int num2zeroLength;
int subCompare;
int cnt;
int switchNum2[SIZE] = { 0 };

for (int i = 0; i < SIZE; i++) {
    temp[i] = reNum1[i];
}
```

Turn the index over, and define the temp array for division

```
for (int i = 0; i < SIZE; i++) {
    cnt = 0;

    while (1) {

        tempzeroLength = SIZE - 1;
        num2zeroLength = SIZE - 1;
        while (temp[tempzeroLength] == 0) { tempzeroLength--; }
        while (reNum2[num2zeroLength] == 0) { num2zeroLength--; }

        if (tempzeroLength == num2zeroLength) {
            subCompare = tempzeroLength;
            while (temp[subCompare] >= reNum2[subCompare]) {
                subCompare--;
            }
            if (subCompare < tempzeroLength) {}
            else { break; }
        }
        else if (tempzeroLength < num2zeroLength) { break; }
```

Compare the two numbers to make sure subtraction is possible

Division implementation

```
for (int n = 0; n <= SIZE - 1; n++) {
    subRes[n] = temp[n] - reNum2[n];
}
for (int n = 0; n <= SIZE - 1; n++) {
    if (n != SIZE - 1 && subRes[n] < 0) {
        subRes[n + 1] -= 1;
        subRes[n] += 10;
    }
}

int zero = 0;

if (subRes[SIZE - 1] < 0) {
    break;
}
for (int n = 0; n < SIZE; n++) {
    temp[n] = subRes[n];
    if (temp[n] == 0) {
        zero += 1;
    }
}
if (zero == SIZE) {
    cnt += 1;
    break;
}
else {
    cnt += 1;
}
```

Repeat subtraction

```
int zero = 0;

divRes[i] = cnt;
for (int j = 1; j < SIZE; j++) {
    reNum2[j - 1] = reNum2[j];
}
if (reNum2[SIZE - 1] != 0) {
    reNum2[SIZE - 1] = 0;
}
for (int n = 0; n < SIZE; n++) {
    if (reNum2[n] == 0) {
        zero += 1;
    }
}
if (zero == SIZE) {
    cnt += 1;
    break;
}
else {
    cnt += 1;
}
```

When the subtraction is completed, add 1 to the cnt to represent the quotient, and stop the division(subtraction) if the temp is 0

Division implementation

Output the result

```c
int size = SIZE - 1;
int zeroNum = 0;
printf("Result (div) : ");
while (divRes[size] == 0) {
    zeroNum += 1;
    size--;
}
int existNum = SIZE - zeroNum;
if (resSign < 0) {
    printf("-");
}
printf("%d", divRes[0]);
printf(".");
for (int i = 1; i < existNum; i++)
    printf("%d", divRes[i]);
```

Minus sign output & Adding a decimal point

```
> Executing task: cmd /C 'd:\Computer Systems\c언어\division' <

Please enter the floating number : 12121212121212121212121212.1212121212121212
Please enter the floating number : 98989898989898989898989898.989898989898989898989898
Result (div) : 0.00122448979591836734693877551020408163265091836734816326530612244897959183673469387755101826530614
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

> Executing task: cmd /C 'd:\Computer Systems\c언어\division' <

Please enter the floating number : 444444.44444
Please enter the floating number : 1111.111
Result (div) : 400.000039996003999600399960039996003999600399960039996003999600399960039996003999600399960039996003999600399960042
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

The result of calculating with following numbers and random numbers

Thank you

For Listening