Project #1

2021 2-2 Computer Systems

18102080 Shin Dongwoo
20102117 Lee Jeonghyeon
20102125 Pi Yujin

Team 6

Contents

- Largest Floating Number
- Exceed the Maximum range
- Operation
  - Addition
  - Subtraction
  - Multiplication
  - Division

1 Largest Floating Number

```
float minimum number : 340282346638528859811704183484516925440.000000
float minimum number : 0.000000
double maximum number : 1.797693e+308
double minimum number : 2.225074e-308
```

- Float
  - Minimum: 1.175494351 E - 38
  - Maximum: 3.402823466 E + 38
- Double
  - Minimum:

    2.2250738585072014 E - 308
  - Maximum:

    1.7976931348623158 E + 308

1 Largest Floating Number

```
int main()
{
    char s1[] = "+1797693134862315708145274237317043567980705675258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658013493321880858593 89";
    char s1_1[100] = "+1797693134862315708145274237317043567980705675258449965989174768031572607800285387605895586327668781715404589535";
```

```
printf("s1: %s\n", s1);
```

1797693134862315708145274237317043567980705675258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658013493321880858593 89

- Using Array
  - Adjust the length of the arrangement
  - Can express numbers of any length
- Let ‘s add the maximum value of the double.

1 Largest Floating Number

```
s1: 11797693134862315708145274237317043567980705675258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684675467035375169
86049910576551282076245490090389328944075868508455133942304583236903222948165801349332188085859389
```

*Can represent double type **maximum number**!*

2 Exceed the Maximum range

```
int main()
{
    char s1[] = "+9898989898989898989898989898989898989898989898989898";//"-98989898989898989898989898.98989898989898989898";
    char s1_1[100] = "+98989898989898989898989898989898989898989898989898";//"+9898989898989898989898989898989898989898989898989898";


    char s2[] = "+12121212121212121212121212121212121212121212"; //"+121212121212121212121212.1212121212121212";
    char s2_1[100] = "+12121212121212121212121212121212121212121212"; //"+121212121212121212121212.1212121212121212";
```

```
s1_1 without plus or minus sign: 9898989898989898989898989898989898989898989898
s2_1 without plus or minus sign: 12121212121212121212121212121212121212121212
s1: +9898989898989898989898989898989898989898989898
size of s1: 48
s2: +12121212121212121212121212121212121212121212
```

- Output a number in the range that exceeds the maximum value.
- 12121212121212121212121212. 1212121212121212
- 98989898989898989898989898. 9898989898989898989898

3 Operations

- Addition
- Subtraction
- Multiplication
- Division

Addition

```
void eliminate(char* str, char ch) {
    for (; *str != '\0'; str++) {
        if (*str == ch) {
            strcpy_s(str, 101, str + 1);
            str--;
        }
    }
}
```

```
int search(char ch, int n, char str[])
{
    int idx = -1;
    for (int i = 0; i < n; i++) {
        if (str[i] == ch) idx = i;
    }
    return idx + 1;
}
```

- Check the positive and negative symbols and erase them using the ‘eliminate’ function for convenience of calculation later.
- Use the 'search' function to locate the decimal point of two numbers.

Addition

```
int sn1 = sizeof(n1);
sprintf_s(n1, sn1, "%099s", s1_1);

for (int i = ss2 - 1; i >= ss2 - (ss1 - ss2 - gap); i--)
{
    append('0', s2);
}

int sn2 = sizeof(n2);

sprintf_s(n2, sn2, "%099s", s2_1);
```

- Add 0 to the back of the short number by the difference between the two decimal points to fit it to the same position.
- Put both numbers back to the end of the arrangement to make it easier to calculate.

Addition

```
for (int i = sn1 - 2; i >= 0; i--)
{
    if (n1[i] == '.') {
        s3[i + 1] = n1[i];
        continue;
    }
    sum = (n1[i] - '0') + (n2[i] - '0') + carry;
    carry = sum / 10;
    sum = sum % 10;
    s3[i + 1] = sum + '0';
}
s3[0] = carry + '0';

for (i = 0; s3[i] == '0'; i++);
printf("%s\n", s3 + i);
```

- Divide it into one digit and calculate it, and if it exceeds 10, you can add it to the previous number using the carry variable.
- Change this result value back to a character and add it to the s3 result value Array.
- Use repetition to erase the zeros stored in front of the array.
- Outputs from the index of the result value.

Subtraction

```
if (idx2 > idx1) {

    for (int i = sizeof(n2) - 2; i >= 0; i--)
    {
        if (n2[i] == '.') {
            s3[i + 1] = '.';
            continue;
        }
        else {
            int temp1 = n1[i] - '0';
            int temp2 = n2[i] - '0';
            temp2 = temp2 - carry;
            if (temp1 > temp2) {
                carry = 1;
                sub = 10 - temp1 + temp2;
                s3[i + 1] = itoc(sub);
            }
            else {
                sub = temp2 - temp1;
                s3[i + 1] = itoc(sub);
                carry = 0;
            }
        }
    }
    s3[0] = itoc(carry);
    printf("-");
}
```

```
else {
    for (int i = sizeof(n2) - 2; i >= 0; i--)
    {
        if (n2[i] == '.') {
            s3[i + 1] = '.';
            continue;
        }
        else {
            int temp1 = n1[i] - '0';
            int temp2 = n2[i] - '0';
            temp2 = temp2 - carry;
            if (temp1 > temp2) {
                carry = 1;
                sub = 10 - temp1 + temp2;
                s3[i + 1] = itoc(sub);
            }
            else {
                sub = temp2 - temp1;
                s3[i + 1] = itoc(sub);
                carry = 0;
            }
        }
    }
    s3[0] = itoc(carry);
    printf("-");
}
```

- If the number of decimals was different, we divided the cases and subtracted them.

Subtraction

```
else if (idx1 == idx2) {
    if (strlen(s1) >= strlen(s2)) {
        for (int i = sizeof(n1) - 2; i >= 0; i--)
        {
            if (n1[i] == '.') {
                s3[i + 1] = '.';
                continue;
            }
            else {
                int temp1 = n1[i] - '0';
                int temp2 = n2[i] - '0';

                temp1 = temp1 - carry;

                if (temp2 > temp1) {
                    carry = 1;
                    sub = 10 - temp2 + temp1;
                    s3[i + 1] = itoc(sub);
                }
                else {
                    sub = temp1 - temp2;
                    s3[i + 1] = itoc(sub);
                    carry = 0;
                }
            }
        }
        s3[0] = itoc(carry);
    }
```

- In particular, if the decimal place numbers are the same as integers, it is implemented to compare the size of the actual number (strlen (Array) and calculate by dividing the number of cases.

Addition & Subtraction

```
if (operator=='+') {
    if (s1[0] == '+' && s2[0] == '+') {
        Addition(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else if (s1[0] == '+' && s2[0] == '-') {
        Subtraction(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else if (s1[0] == '-' && s2[0] == '+') {
        printf("-");
        Subtraction(s1, s2, s1_1, s2_1, ss1, ss2);
        printf("if \'--\', consider \'--\' as \'+\'");
    }
    else if (s1[0] == '-' && s2[0] == '-') {
        printf("-");
        Addition(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else {
        printf("undefined number");
    }
}
```

```
else if (operator=='-') {
    if (s1[0] == '+' && s2[0] == '+') {
        Subtraction(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else if (s1[0] == '+' && s2[0] == '-') {
        Addition(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else if (s1[0] == '-' && s2[0] == '+') {
        printf("-");
        Addition(s1, s2, s1_1, s2_1, ss1, ss2);
    }
    else if (s1[0] == '-' && s2[0] == '-') {
        printf("-");
        Subtraction(s1, s2, s1_1, s2_1, ss1, ss2);
        printf("if \'--\', consider \'--\' as \'+\'");
    }
    else {
        printf("undefined number");
    }
}
printf("if there is no number, value is 0");
```

- Addition and subtraction can be used freely depending on the sign.
- According to the sign and operation, we organize the cases of addition and subtraction.

Addition & **S**ubtraction

```
s1_1 without plus or minus sign: 9898989898989898989898989898.9898989898989898989898
s2_1 without plus or minus sign: 12121212121212121212121212.1212121212121212
s1: +9898989898989898989898989898.9898989898989898989898
size of s1: 49
s2: +12121212121212121212121212.1212121212121212
size of s2: 41
idx1: 27
idx2: 25
gap: 2
n1: 0000000000000000000000000000000000000000000000000000009898989898989898989898989898.9898989898989898989898
n2: 0000000000000000000000000000000000000000000000000000012121212121212121212121212.1212121212121212000000
input operator: +
991111111111111111111111111.1111111111111110989898
if there is no number, value is 0
```

```
s1_1 without plus or minus sign: 9898989898989898989898989898.9898989898989898989898
s2_1 without plus or minus sign: 12121212121212121212121212.1212121212121212
s1: +9898989898989898989898989898.9898989898989898989898
size of s1: 49
s2: +12121212121212121212121212.1212121212121212
size of s2: 41
idx1: 27
idx2: 25
gap: 2
n1: 0000000000000000000000000000000000000000000000000000009898989898989898989898989898.9898989898989898989898
n2: 0000000000000000000000000000000000000000000000000000012121212121212121212121212.1212121212121212000000
input operator: -
988686868686868686868686686.8686868686868686989898
if there is no number, value is 0
```

Addition & **S**ubtraction

```
s1_1 without plus or minus sign: 12121212121212121212121212.1212121212121212
s2_1 without plus or minus sign: 98989898989898989898989898.9898989898989898989898
s1: +12121212121212121212121212.1212121212121212
size of s1: 41
s2: -98989898989898989898989898.9898989898989898989898
size of s2: 49
idx1: 25
idx2: 27
gap: 2
n1: 00000000000000000000000000000000000000000000000012121212121212121212121212.1212121212121212000000
n2: 00000000000000000000000000000000000000000000000098989898989898989898989898.9898989898989898989898
input operator: +
-988686868686868686868686686.868686868686868686989898
if there is no number, value is 0
```

```
s1_1 without plus or minus sign: 98989898989898989898989898.9898989898989898989898
s2_1 without plus or minus sign: 12121212121212121212121212.1212121212121212
s1: -98989898989898989898989898.9898989898989898989898
size of s1: 49
s2: +12121212121212121212121212.1212121212121212
size of s2: 41
idx1: 27
idx2: 25
gap: 2
n1: 00000000000000000000000000000000000000000000000098989898989898989898989898.9898989898989898989898
n2: 00000000000000000000000000000000000000000000000012121212121212121212121212.1212121212121212000000
input operator: +
-988686868686868686868686686.868686868686868686989898
if '--', consider '--' as '+'if there is no number, value is 0
```

Multiplication

```
for(int i=strlen(s1)-1; i>=1; i--){
        int temp1 = ctoi(s1[i]);
    for(int j=strlen(s2)-1; j>=1; j--){
        int temp2 = ctoi(s2[j]);
        multi = temp1*temp2;
        result[MAX_ARRAY -strlen(s1)+i -strlen(s2)+j] += multi;
        while(1) {
            if(result[MAX_ARRAY -strlen(s1)+i -strlen(s2)+j] >= 10){
            result[MAX_ARRAY -strlen(s1)+i -strlen(s2)+j] -= 10;
            result[MAX_ARRAY -strlen(s1)+i -strlen(s2)+j -1] += 1;
            }
            else {
                break;
            }
        }
    }
}
```

Multiplication

```
//한 숫자가 0일 경우
//todo: 예외처리
int slen1=strlen(s1);
int slen2=strlen(s2);
if (slen1==1 || slen2==1) {
    if(s1[0]=='0' || s2[0]=='0') {
        printf("result: 0₩n");
        return 0;
    }
}
```

- Exception if one number is zero

Multiplication

```
// 부호 판별
if((s1[0] == '-' && s2[0] == '-') || (s1[0] != '-' && s2[0] != '-' ))
    printf("+");
else {
    printf("-");
}


//소수점 삽입 후 프린트
for(int i=0; i<MAX_ARRAY; i++) {
    if (MAX_ARRAY-i == point) {
        printf(".");
        printf("%d", result[i]);
    }
    else {
        printf("%d", result[i]);
    }
}
```

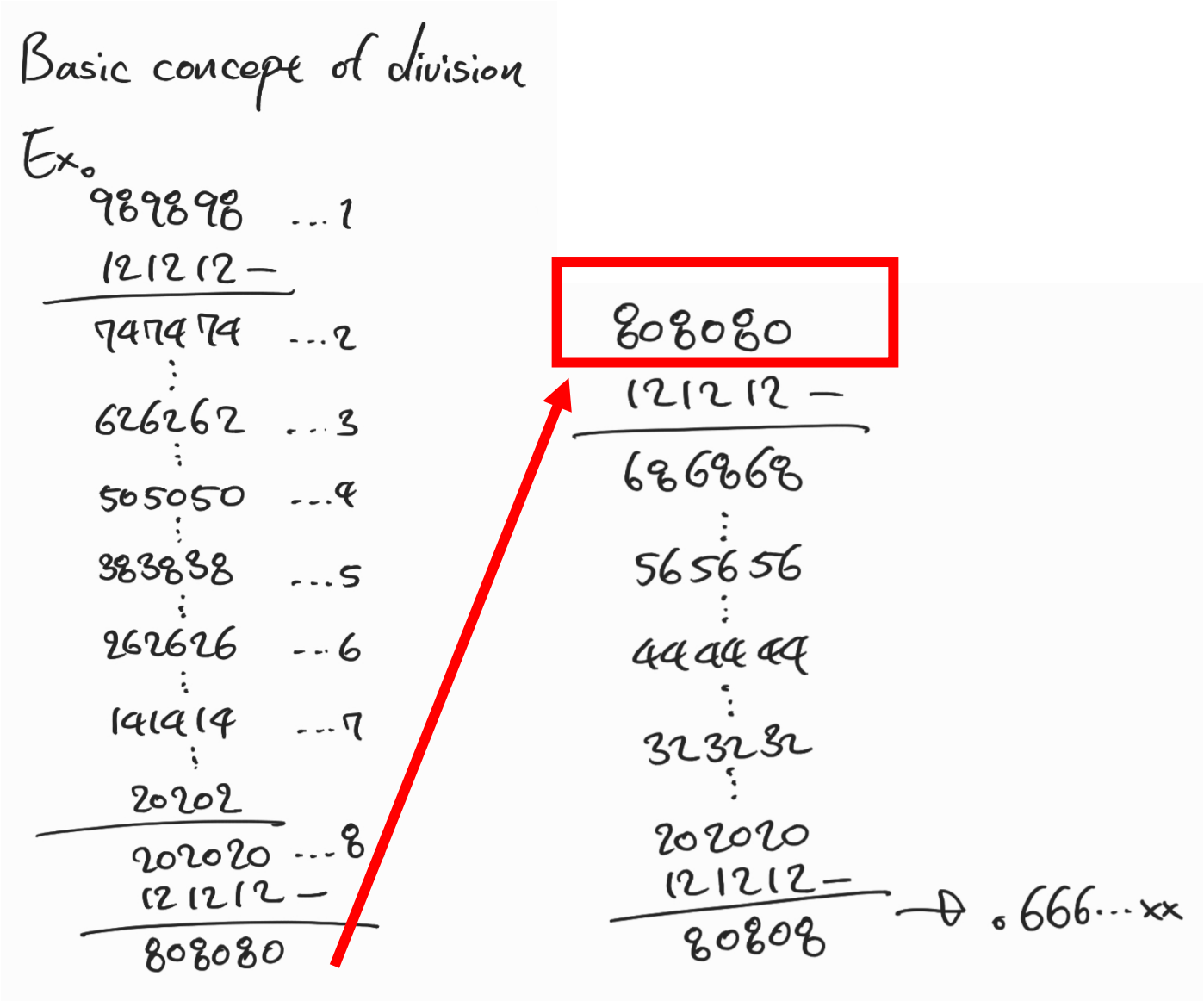
- It determines the sign of two numbers and outputs the sign
- Remember the decimal place of the original number and output it

Multiplication

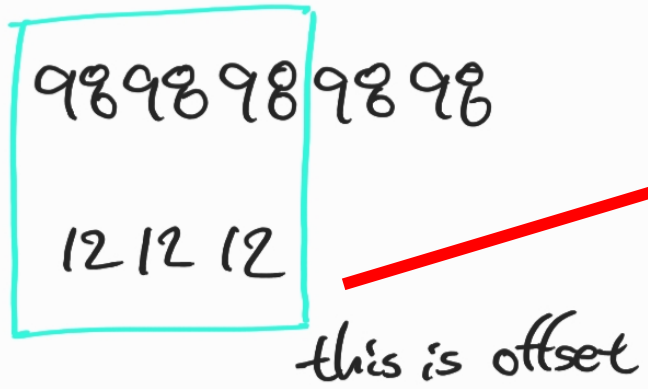
```
s1: +98989898989898989898989898.9898989898989898989898
s2: +121212121212121212121212.1212121212121212
idx1: 21
idx2: 15
point: 37
s1 without point: +989898989898989898989898989898989898989898989898
s2 without point: +1212121212121212121212121212121212121212
+98989898989898989898989898.9898989898989898989898 * +121212121212121212121212.1212121212121212 = +000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000119987756351392715029078665442
3018059381575757574557.6981940618304254667891031527395163760
```

```
s1: +98989898989898989898989898.9898989898989898989898
s2: 0
idx1: 21
point: 22
s1 without point: +989898989898989898989898989898989898989898989898
s2 without point: 0
result: 0
```

Division


Basic concept of division
Ex.
989898 ...1
121212-
747474 ...2
626262 ...3
505050 ...4
383838 ...5
262626 ...6
141414 ...7
20202
202020 ...8
121212-
808080
808080
121212-
686868
565656
444444
323232
202020
121212-
80808
→ .666...xx

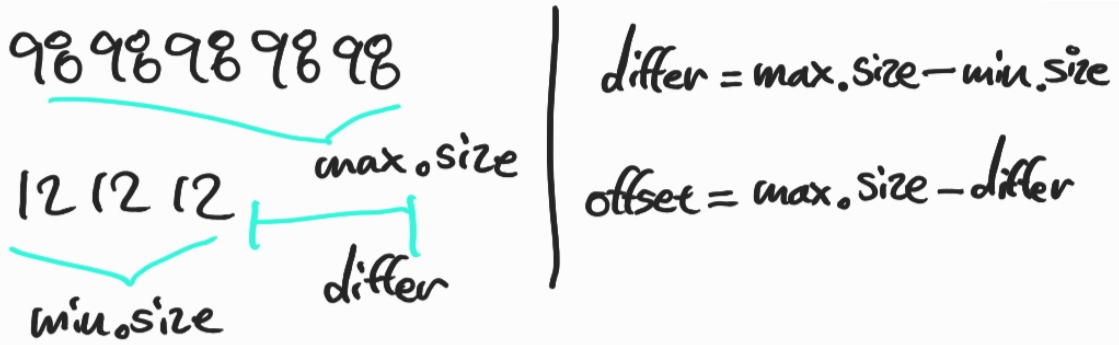
Division



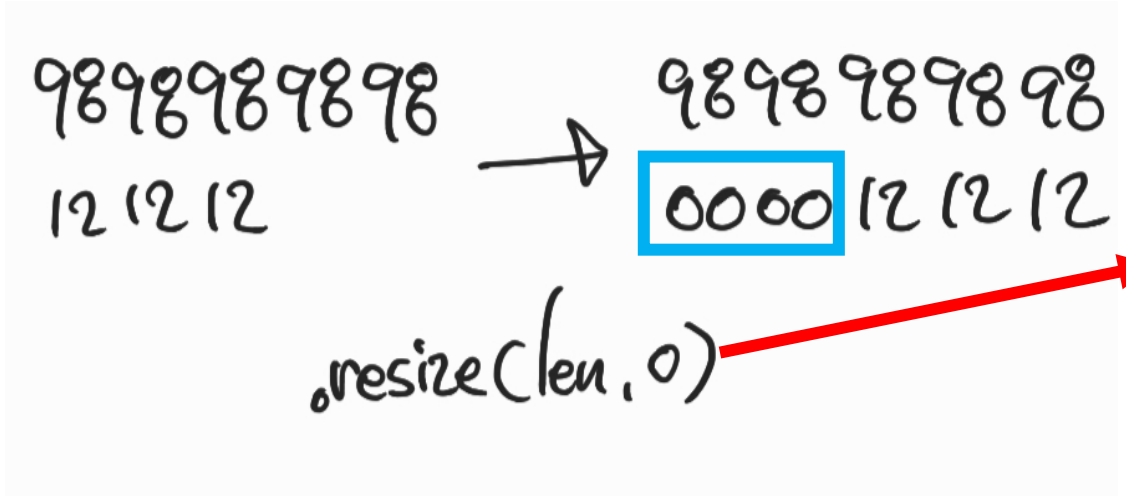

```
std::size_t len = std::max(dividend.num.size(), divisor.num.size());
std::size_t diff = std::max(dividend.num.size(), divisor.num.size()) -
std::size_t offset = len - diff - 1;
```

```
dividend.num.resize(len, '0');
divisor.num.resize(len, '0');
quotient.num.resize(len, '0');
```

```
memmove(&divisor.num[diff], &divisor.num[0], len - diff);
memset(&divisor.num[0], '0', diff);
```



Division

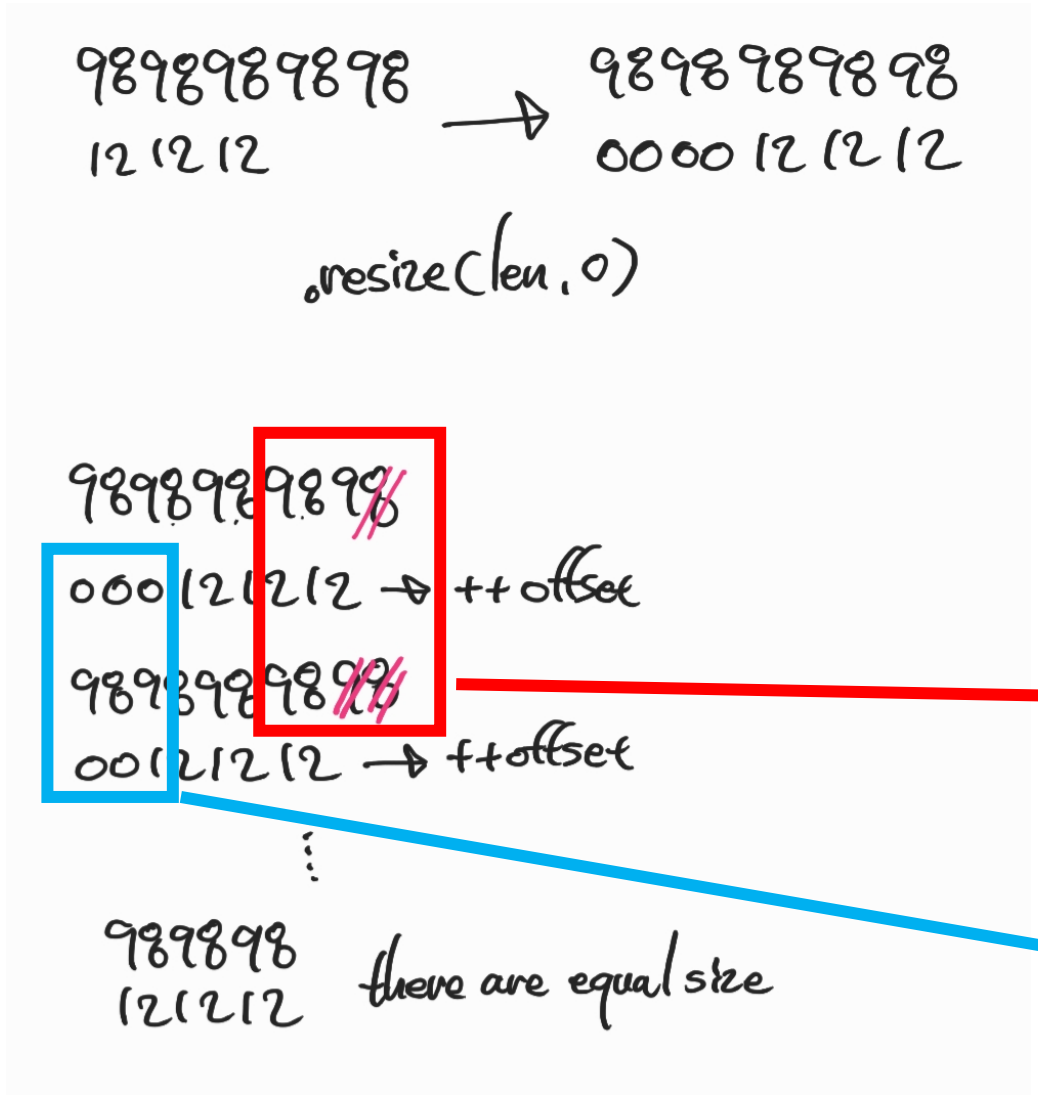



```
std::size_t len = std::max(dividend.num.size(), divisor.num.size());
std::size_t diff = std::max(dividend.num.size(), divisor.num.size()) -
std::size_t offset = len - diff - 1;
```

```
dividend.num.resize(len, '0');
divisor.num.resize(len, '0');
quotient.num.resize(len, '0');
```

```
memmove(&divisor.num[diff], &divisor.num[0], len - diff);
memset(&divisor.num[0], '0', diff);
```

Division

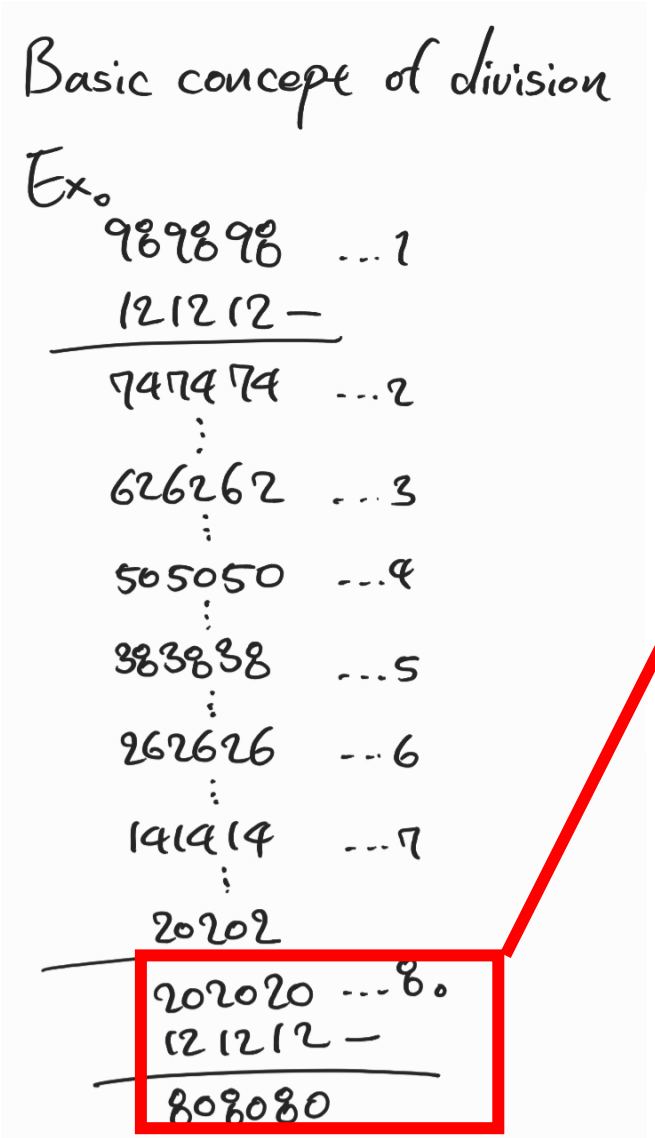



```
void LargeInt::norma(){
    for (int I = num.size() - 1; I >= 0; --I) {
        if (num[I] != '0')
            break;
        num.erase(I, 1);
    }
    if (num.empty()){
        num = "0";
        sign = '~';
    }
}
```

```
    if (remaining && offset == len - 1){
        remaining->num = dividend.num;
        remaining->sign = rem_sign;
        remaining->norma();
        if (remaining == this){
            return *this;
        }
    }
```

```
    memmove(&divisor.num[0], &divisor.num[1], len - 1);
    memset(&divisor.num[len - 1], '0', 1);
    ++offset;
}
```

Division




```
while(offset < len){
    while (dividend >= divisor){
        int borrow = 0, total = 0;
        for (std::size_t I = 0; I < len; ++I){
            total = dividend.indexing(I) - divisor.indexing(I) - borrow;
            borrow = 0;
            if (total < 0){
                borrow = 1;
                total += 10;
            }
```

Division ⟨float & int⟩

Success #stdin #stdout 0s 5592KB

stdin

```
Standard input is empty
```

stdout

```
idx1: 22
idx2: 16
point: 6
s1 without point: 9898989898989898989898989898998989898988989898
s2 without point: 12121212121212121212121212121212121212
+816.66666
```

Success #stdin #stdout 0s 5580KB

stdin

```
Standard input is empty
```

stdout

```
idx1: 22
idx2: 16
point: 6
s1 without point: 60060060060060060060060060060060060060060060060
s2 without point: 40404040404040404040404040404040404040
+148.48648
```

Success #stdin #stdout 0.01s 5300KB

stdin

```
Standard input is empty
```

stdout

```
idx1: 25
idx2: 21
point: 4
s1 without point: 8877733322211122233377788
s2 without point: 333222111777111222333
+26642
```

Project #1

2021 2-2 Computer Systems

Thank you =)