Software Assisted Arithmetic Operations

- Multiplication and Division -

Prepared by

Lee Chang Yoon

32183641@dankook.ac.kr

October 5, 2021

Executive Summary

```
leechangyoon@                    :~/system_programming$ tree week6
week6
├── big_integer
└── big_integer.c

0 directories, 2 files
```

This report is for homework 3 of the system programming class. The main goal of this report is to build a C program that calculates 100000 * 100000. The following requirements: do not use long long type, do not use double type, produce an accurate answer, and present that answer in a hexadecimal number. To solve this problem, we will use the string data to satisfy the following requirements. We will explain more about multiplication in section 1. Also, to present the decimal number into hexadecimal numbers, we implemented conversion by subtraction and division. We will explain more details about the subtraction in section 3 and division in section 4. The source code is at the end of the report.

1. Multiplication

```
int alen = strlen(astr);
int blen = strlen(bstr);
// Multiplication
mul(astr, alen, bstr, blen, res);
```

Figure 1-1

```
// Calculation
for (i = 0; i < an; i++)
        for (j = 0; j < bn; j++)
                res[i + j] += A[an - i - 1] * B[bn - j - 1];

// Raise the exceeded numbers for each digits.
for (i = 0; i < an + bn; i++) {
        tmp = res[i] / 10;
        res[i] = res[i] % 10;
        res[i + 1] = res[i + 1] + tmp;
}
```

Figure 1-2

The main requirement of the program is to multiply two big integers. We implemented this operation in the multiplication function. In Figure 1-1, we can check that each decimal number is stored as string type data in variable astr and bstr. The multiplication function takes variables ap and bp, which are the pointers of two numbers that are to be calculated. Also, it takes an and bn, which are the length of each string of ap and bp. After operating multiplication for two decimal numbers, it stores the result into the res array. Let's see Figure 1-2. Figure 1-2 is part of the multiplication. First, it multiplies each digit, from last to first, and stores each result into the res array. Then, for each digit, if there is an exceeding number, raise it to the next index of the res array.

2. Subtraction

```
// Preprocessing
for (int i = len - 1; i >= 0; i--) {
        r[i] -= n[i];
        if (r[i] < 0) {
                r[i] += 10;
                r[i - 1]--;
        }
}
```

Figure 2

Before we get into the division, let's see subtraction that is used to implement the division function. Due to the data type for the following decimal numbers being a string, we implemented a dividing operation by subtraction. Let's denote the divider as 16, which is the number for hexadecimal number. Figure 2 is the main part of the subtraction function. Check out Figure 2. This function will subtract 16 for each digit, from first to last. If the result of the subtraction is less than 0, then it makes the data of the current index to 0 and subtracts 1 to the data of the next index. This is the basic calculation that we do in real life. By repeating this operation, we can implement division.

3. Division

```
// Subtraction -> Preprocessing
for (i = 0; i <= rn - nn; i++) {
        while (sub(R + i, N, nn)) tmp[i]++;
        R[i + 1] += R[i] * 10;
}
```

Figure 3-1

```
// Division
while (rlen > 0) {
        cur[0] = hex[div(res, rlen, not, nlen)];
        strcat(ans, cur);
        rlen = strlen(res);
}
```

Figure 3-2

Now, let's see how division works in this code. Figure 3-1 is the main part of the division code. As we mentioned in section 2, the operation of dividing number by divider, we implemented by subtraction. According to the Figure 3-1, it subtracts the following number until it reaches to the last digit of the number. After all of the subtractions, remainder is left in the last index of the array, and this remainder becomes hexadecimal digit. In short, this function returns following hexadecimal digit and stores quotient in ap, which was array of the target number that we divided. The main function uses this division function in Figure 3-2. Let's check Figure 3-2. In Figure 3-2, it divides the target number until it gets smaller than 0. Also, for each loop statement, it stores each hexadecimal digit into the ans array. As a result, we can get the hexadecimal number from decimal number by using this division function.

Conclusion

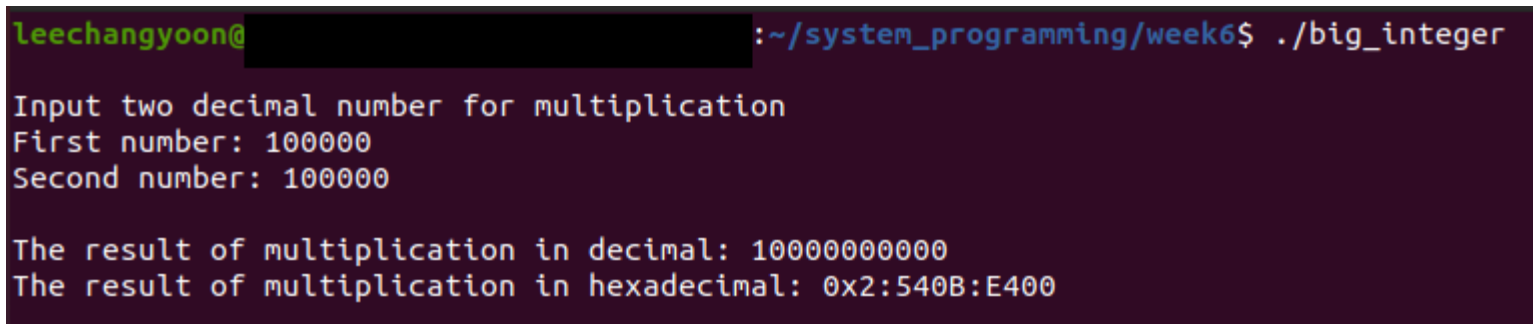

Figure 4-1

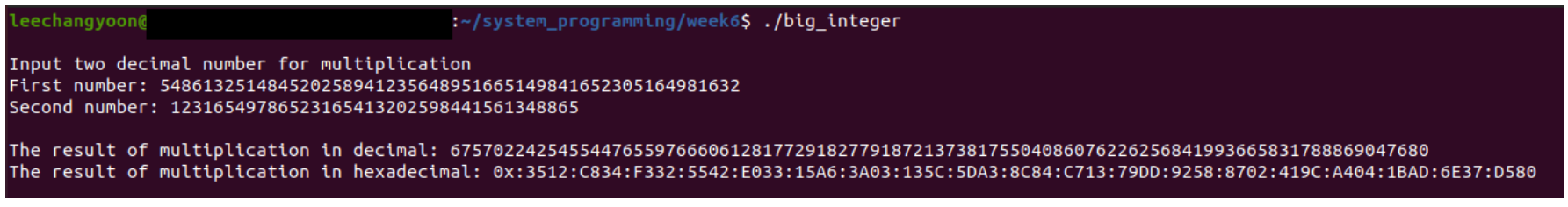

Figure 4-2

By using multiplication, subtraction and division, we can build the required C program that calculates 100000 * 100000. This is the program that we did not use the supporting libraries. Now days, there are already a lot of libraries that supports the calculation of big integers. For example, JAVA supports BigInteger class. Also, C++ has Big Integer Library. Even we can use those libraries, we implemented program with the simple string data type. The result of the program is in Figure 4-1 and 4-2. The source code of the program is appended in the last section.

Appendix

```c
//////////////////////////////
// * Name: big_integer.c     //
// * Author: Lee Chang Yoon  //
// * ID: 32183641        //
// * Date: 2021.10.04        //
//////////////////////////////

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


#define MAX 201
char hex[16] = {'0', '1', '2', '3', '4', '5', '6', '7',
                '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};


void    encode(char* s, int len, int num[]);
int     sub(int* r, int* n, int len);
void    mul(char* ap, int an, char* bp, int bn, char* rp);
int     div(char* rp, int rn, char* n ,int nn);


int main() {
    char astr[MAX];
    char bstr[MAX];
    char res[MAX] = "";

    printf("\nInput two decimal number for multiplication");
    printf("\nFirst number: ");
    scanf("%s", astr);
    printf("Second number: ");
    scanf("%s", bstr);

    char not[] = "16";
    char ans[MAX] = "";
    // strcat only take the variables that is char*.
    // Therefore, we have to put the blank character at the end of the string.
    char cur[2] = " \0";

    int alen = strlen(astr);
    int blen = strlen(bstr);
    // Multiplication
    mul(astr, alen, bstr, blen, res);

    int rlen = strlen(res);
    int nlen = strlen(not);
    printf("\nThe result of multiplication in decimal: ");
    for (int i = 0; i < rlen; i++) printf("%c", res[i]);

    // Division
    while (rlen > 0) {
        cur[0] = hex[div(res, rlen, not, nlen)];
        strcat(ans, cur);
        rlen = strlen(res);
    }

    printf("\nThe result of multiplication in hexadecimal: 0x");
    for (int i = strlen(ans) - 1; i >= 0; i--) {
        if (i % 4 == 3) printf(":");
        printf("%c", ans[i]);
    }
```

```
    printf("\n\n");
    return 0;
}

////////////////////////////// encode ////////////////////////////////
// Data size of the big integer is too big.                         //
// Therefor, we have to encode the following big integer to string. //
//////////////////////////////////////////////////////////////////////

void encode(char* s, int len, int num[]) {
    for (int i = 0; i < len; i++)
        num[i] = s[i] - '0';
}


/////////////////////////////////////////////// subtraction ///////////////////////////////////////////////
/////
// Without using the library which supports big integer, we have to implement division with using subtractio
n. //
// We will preprocess the subtraction for each digit and build the result in the dividion function.
   //
////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////

int sub(int* r, int* n, int len) {
    for (int i = 0; i < len; i++) {
        if (r[i] > n[i]) break;
        if (r[i] < n[i]) return 0;
    }

    // Preprocessing
    for (int i = len - 1; i >= 0; i--) {
        r[i] -= n[i];
        if (r[i] < 0) {
            r[i] += 10;
            r[i - 1]--;
        }
    }
    return 1;
}


////////////////////////// multiplication ////////////////////////////////////////////////////
// First, we will encode the each number that we are going to multiply.                     //
// Next, we will store the result of multiplication of each digit to the res array.         //
// Then, we have to process the cacluation on raising the exceeded numbers for each digits. //
// In short, we will store the result of the multiplication to the following array.         //
//////////////////////////////////////////////////////////////////////////////////////////////

void mul(char* ap, int an, char* bp, int bn, char* rp) {
    int i, j, tmp;
    int A[MAX], B[MAX];
    int res[MAX] = {0};

    // Encoding
    encode(ap, an, A);
    encode(bp, bn, B);

    // Calculation
    for (i = 0; i < an; i++)
        for (j = 0; j < bn; j++)
            res[i + j] += A[an - i - 1] * B[bn - j - 1];

    // Raise the exceeded numbers for each digits.
    for (i = 0; i < an + bn; i++) {
```

```
        tmp = res[i] / 10;
        res[i] = res[i] % 10;
        res[i + 1] = res[i + 1] + tmp;
    }

    // Find the index of top digit
    for (i = an + bn; i >= 0; i--)
        if (res[i] > 0) break;

    // Store the data in the following array.
    j = i;
    for (; i >= 0; i--)
        rp[j - i] = (char)(res[i] + '0');
}


//////////////////////////// division ////////////////////////////////////////////////////////////
/
// According to the remaek in 'subtraction', computer itself does not support the division to big integer. /
/
// Therefore, we have to subtract the target notation untill it reaches to the last digit.                 /
/
// After each div function operations, the remainder is stored in the last digit of the target number.     /
/
// By using following remainder, we can get the notated(hex) number of each digits from last to first.     /
/
// This function returns the each notated(hex) digit.                                                      /
/
////////////////////////////////////////////////////////////////////////////////////////////////////////
/

int div(char* rp, int rn, char* n ,int nn) {
    int i, flag;
    int R[MAX] = {0};
    int N[MAX] = {0};
    int tmp[MAX] = {0};

    // Encoding
    encode(rp, rn, R);
    encode(n, nn, N);

    // Subtraction -> Preprocessing
    for (i = 0; i <= rn - nn; i++) {
        while (sub(R + i, N, nn)) tmp[i]++;
        R[i + 1] += R[i] * 10;
    }

    // Check if there is the first digit
    // If first digit remains after division, we will record the data from the first digit.
    // Else, record from the second digit.
    if (tmp[0]) flag = 0;
    else        flag = -1;

    // Record the data
    for (i = rn; i >= rn - nn; i--) rp[i] = 0;
    for (i = 0; i <= rn - nn; i++)  rp[i + flag] = (char)(tmp[i] + '0');

    return R[rn - 1];
}
```