

# Домашнє завдання №18

Написати на C/C++ реалізацію довгих чисел на основі списку(елементи списку – десяткові цифри) з використанням вказівників. Реалізувати дію згідно варіанту.

## Вибір варіанту

$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 2 + 1$$

де:  $N_{\text{ж}}$  – порядковий номер студента в групі, а  $N_{\text{г}}$  – номер групи(1,2,3,4,5,6,7 або 8)

## Варіанти завдань

Варіант	Розмір масиву для реалізації хеш-таблиці
1	$c = a + b$ , де $a$ – довге число, $b$ – довге число, $c$ – довге число
2	$c = a - b$ , де $a$ – довге число, $b$ – довге число, $c$ – довге число

## Приклад коду

Лістинг

```
#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

#define ADDON_DIGITS 2
#define MAX_STR_LEN 4096
#define MAX_MSG_LENGTH 1024
class BigValue;
struct BigValueVariableInput {
    char msg[MAX_MSG_LENGTH];
    BigValue*& bigValueVariablePtr;
    char bigValueStr[MAX_STR_LEN];
};

struct Digit {
    unsigned char value;
    struct Digit* prevDigit;
    struct Digit* nextDigit;
};

class BigValue {
private:
    Digit* msb;
    Digit* lsb;
    bool carry;
public:
    BigValue(char* str, unsigned int addonDigitsCount = ADDON_DIGITS) {
        msb = NULL;
        lsb = NULL;
        if (!verifyStr(str)) return;
        Digit* digitPtr = NULL;
        Digit* prevDigitPtr = NULL;
        for (char* endStr = str + strlen(str) - 1; str <= endStr; --endStr) {
```

```

        if (!(digitPtr = (Digit*)malloc(sizeof(Digit)))) break;
        digitPtr->value = *endStr - '0';
        digitPtr->prevDigit = prevDigitPtr;
        prevDigitPtr ? (prevDigitPtr->nextDigit = digitPtr) : (lsb =
digitPtr);
        prevDigitPtr = digitPtr;
    }

    while (addonDigitsCount-- > 0) {
        if (!(digitPtr = (Digit*)malloc(sizeof(Digit)))) break;
        digitPtr->value = 0;
        digitPtr->prevDigit = prevDigitPtr;
        prevDigitPtr ? (prevDigitPtr->nextDigit = digitPtr) : (lsb =
digitPtr);
        prevDigitPtr = digitPtr;
    }

    (digitPtr) ? (digitPtr->nextDigit = NULL) : (0);
    msb = digitPtr;

}

static void initAMinusB(BigValue& a, BigValue& b) {
    b.complement10();
}

static void initBMinusA(BigValue& a, BigValue& b) {
    a.complement10();
}

static void addDigit(unsigned char& c, unsigned char& a, unsigned char& b,
unsigned char& carry) {
    c = a + b + carry;
    carry = c / 10;
    c %= 10;
}

static void andDigit(unsigned char& c, unsigned char& a, unsigned char& b,
unsigned char& carry) {
    c = a & b;
}

static void orDigit(unsigned char& c, unsigned char& a, unsigned char& b, unsigned
char& carry) {
    c = a | b;
}

static void xorDigit(unsigned char& c, unsigned char& a, unsigned char& b,
unsigned char& carry) {
    c = a ^ b;
}

BigValue(BigValue& a, BigValue& b, void(*prepareBigValue)(BigValue& a, BigValue&
b), void(*opDigit)(unsigned char& c, unsigned char& a, unsigned char& b, unsigned char&
carry), unsigned int addonDigitsCount = 0) {
    if (prepareBigValue != NULL) {
        prepareBigValue(a, b);
    }
    Digit* digitPtrA = a.lsb;
    Digit* digitPtrB = b.lsb;
    Digit* digitPtrC = NULL;
    Digit* currDigitPtrC = msb = /**/lsb/**/ = NULL;
    unsigned char valueA, valueB, carryValue = 0;
    while (digitPtrA != NULL || digitPtrB != NULL || carryValue != 0) {
        if (digitPtrA != NULL) {
            valueA = digitPtrA->value;
            digitPtrA = digitPtrA->nextDigit;
        }
        else {
            valueA = 0;
        }
    }
}

```

```

        if (digitPtrB != NULL) {
            valueB = digitPtrB->value;
            digitPtrB = digitPtrB->nextDigit;
        }
        else {
            valueB = 0;
        }

        if (!(digitPtrC = (Digit*)malloc(sizeof(Digit)))) break;
        digitPtrC->prevDigit = currDigitPtrC;
        currDigitPtrC ? (currDigitPtrC->nextDigit = digitPtrC) : (lsb =
digitPtrC);

        currDigitPtrC = digitPtrC;
        digitPtrC = NULL;
        currDigitPtrC->value = 0;

        opDigit(currDigitPtrC->value, valueA, valueB, carryValue);
    }

    (currDigitPtrC) ? (currDigitPtrC->nextDigit = NULL) : (0);
    msb = currDigitPtrC;

    carry = a.carry | b.carry;
    (carry && msb && msb->value == 1) ? (msb->prevDigit ? msb->value = msb-
>prevDigit->value : 0) : (0);

    }
    void inv10() {
        for (Digit* digitPtr = lsb; digitPtr; digitPtr->value = 9 - digitPtr-
>value, digitPtr = digitPtr->nextDigit);
    }
    void complement10() {
        (carry && msb && msb->value == 1) ? (msb->prevDigit ? msb->value = msb-
>prevDigit->value : 0) : (0);
        inv10();
        carry = msb && msb->value != 0;
        increment();
    }
    char getMSBValue() {
        return (msb != NULL) ? msb->value : 0;
    }
    void add(BigValue& b) { // old implementation
        Digit* digitPtrA = lsb;
        Digit* digitPtrB = b.lsb;
        Digit* currDigitPtrA = msb;
        char valueB, carryValue = 0;
        while (digitPtrA != NULL || digitPtrB != NULL || carryValue != 0) {
            if (digitPtrA != NULL) {
                currDigitPtrA = digitPtrA;
                digitPtrA = digitPtrA->nextDigit;
            }
            else {
                if (!(digitPtrA = (Digit*)malloc(sizeof(Digit)))) break;
                digitPtrA->prevDigit = currDigitPtrA;
                currDigitPtrA ? (currDigitPtrA->nextDigit = digitPtrA) : (lsb
= digitPtrA);

                currDigitPtrA = digitPtrA;
                digitPtrA = NULL;
                currDigitPtrA->value = 0;
            }

            if (digitPtrB != NULL) {
                valueB = digitPtrB->value;
                digitPtrB = digitPtrB->nextDigit;
            }

```

```

        else {
            valueB = 0;
        }

        currDigitPtrA->value += valueB + carryValue;
        carryValue = currDigitPtrA->value / 10;
        currDigitPtrA->value %= 10;
    }

    (currDigitPtrA) ? (currDigitPtrA->nextDigit = NULL) : (0);
    msb = currDigitPtrA;

    carry |= b.carry;
    (carry && msb && msb->value == 1) ? (msb->prevDigit ? msb->value = msb-
>prevDigit->value : 0) : (0);
}

void increment() {
    char carryValue = 1;
    Digit* digitPtr = lsb;
    for (; digitPtr; digitPtr = digitPtr->nextDigit) {
        digitPtr->value += carryValue;
        carryValue = digitPtr->value / 10;
        digitPtr->value %= 10;
    }
    Digit* prevDigitPtr = msb;
    if (carryValue) {
        while (carryValue) {
            if (!(digitPtr = (Digit*)malloc(sizeof(Digit)))) break;
            digitPtr->value = carryValue;
            carryValue = digitPtr->value / 10;
            digitPtr->value %= 10;
            digitPtr->prevDigit = prevDigitPtr;
            prevDigitPtr ? (prevDigitPtr->nextDigit = digitPtr) : (lsb =
digitPtr);

            prevDigitPtr = digitPtr;
        }
        (digitPtr) ? (digitPtr->nextDigit = NULL) : (0);
        msb = digitPtr;
    }
}

private: bool verifyStr(char* str) {
    if (str == NULL) return false;
    for (; *str != '\0' && *str >= '0' && *str <= '9'; str++);
    return (*str == '\0');
}

public: static int scan(BigValueVariableInput bigValueVariableInput[], unsigned int
inputsCount, unsigned int addonDigitsCount = ADDON_DIGITS) { // input
    unsigned int maxSize = 0;
    for (unsigned inputIndex = 0; inputIndex < inputsCount; inputIndex++)
    {
        bigValueVariableInput[inputIndex].msg[0] != '\0' ?
puts(bigValueVariableInput[inputIndex].msg) : 0;
        char* inputBuffer =
bigValueVariableInput[inputIndex].bigValueStr;
        inputBuffer[0] = '\0';
        scanf("%s", inputBuffer); //gets(inputBuffer);
        char* inputBuffer_ = (*inputBuffer == '-' || *inputBuffer ==
'+') ? (inputBuffer + 1) : inputBuffer;
        unsigned int currSize = (unsigned int)strlen(inputBuffer_);
        maxSize < currSize ? maxSize = currSize : 0;
    }
    for (unsigned inputIndex = 0; inputIndex < inputsCount; inputIndex++)
    {
        char* inputBuffer =
bigValueVariableInput[inputIndex].bigValueStr;

```

```

        char* inputBuffer_ = (*inputBuffer == '-' || *inputBuffer ==
'+') ? (inputBuffer + 1) : inputBuffer;
        BigValue* bigValue =
bigValueVariableInput[inputIndex].bigValueVariablePtr = new BigValue(inputBuffer_,
maxSize - (unsigned int)strlen(inputBuffer_) + addOnDigitsCount);
        if (!bigValue) {
            return -1;
        }
        if (*inputBuffer == '-') bigValue->complement10();
    }
    return 0;
}

public: void print() { // output
    (getMSBValue() == 9) ? (complement10(), putchar('-')) : 0;
    bool meaningValue = false;
    for (Digit* digitPtr = msb; digitPtr; digitPtr = digitPtr->prevDigit) {
        (digitPtr->value == 0 && !meaningValue) ? (0) : (meaningValue =
true);
        if (meaningValue) putchar(digitPtr->value + '0');
    }
    if (!meaningValue) putchar('0');
}
};

void add_scenario() { // BigValue c = a + b;
    BigValue* a;
    BigValue* b;
    BigValueVariableInput bigValueVariableInput[2] = {
        { "Please, input a:", a },
        { "Please, input b:", b }
    };
    BigValue::scan(bigValueVariableInput, 2);

    BigValue c = BigValue(*a, *b, NULL, BigValue::addDigit);
    c.print();
}

void sub_scenario() { // BigValue c = a - b;
    BigValue* a;
    BigValue* b;
    BigValueVariableInput bigValueVariableInput[2] = {
        { "Please, input a:", a },
        { "Please, input b:", b }
    };
    BigValue::scan(bigValueVariableInput, 2);

    BigValue c = BigValue(*a, *b, BigValue::initAMinusB, BigValue::addDigit);
    c.print();
}

void lessThanOrEqual_scenario() { // bool c = a <= b;
    BigValue* a;
    BigValue* b;
    BigValueVariableInput bigValueVariableInput[2] = {
        { "Please, input a:", a },
        { "Please, input b:", b }
    };
    BigValue::scan(bigValueVariableInput, 2);

    BigValue d = BigValue(*a, *b, BigValue::initBMinusA, BigValue::addDigit);
    bool c = (d.getMSBValue() != 9);

    printf(c ? "true" : "false");
}

int main(int argc, char** argv) {
    //add_scenario(); // BigValue c = a + b;
    //sub_scenario(); // BigValue c = a - b;
}

```

```
        lessThanOrEqual_scenario()); // bool c = a <= b;

#ifdef __linux__
    (void)getchar();
#elif defined(_WIN32)
    system("pause");
#else
#endif

    return 0;
}
```