МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: Куценко Борис Дмитриевич

Группа: М8О-207Б-20

Преподаватель: Дорохов Евгений Павлович

Москва, 2021

Задание:

Разработать программу на языке C++ согласно варианту задания. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

Вариант №13:

Создать класс Long для работы с целыми беззнаковыми числами из 64 бит. Число должно быть представлено двумя полями unsigned int. Должны быть реализованы арифметические операции, присутствующие в C++, и сравнения.

Описание программы:

Исходный код разделён на 3 файла:

- long.h – описание класса long
- long.cpp – реализация класса long
- main.cpp – основная программа

Дневник отладки:

Ничего плохого не произошло.

Вывод:

В данной лабораторной работе я реализовал пользовательские литерал для работы с беззнаковыми числами класса Long. Это очень удобно, так как с помощью него, можно работать с этим классом, как с обычными числами.

Листинг программ:

long.h:

```
#include <iostream>
#include <string>
#include <cassert>
class Long {
    public :
        Long() : low_digits(0), high_digits(0) {}
        Long(std::string number);
        Long(unsigned int first, unsigned int second) : high_digits(first),
low_digits(second) {}
        friend Long operator+(const Long& first, const Long& second);
        friend Long operator-(const Long& first, const Long& second);
        friend Long operator*(const Long& first, const Long& second);
        friend Long operator/(const Long& first, const Long& second);
        friend Long operator%(const Long& first, const Long& second);
        friend bool operator<(const Long& first, const Long& second);
        friend bool operator==(const Long& first, const Long& second);
        friend bool operator>(const Long& first, const Long& second);
        friend bool operator!=(const Long& first, const Long& second);
```

```
        friend std::istream& operator>> (std::istream &in, Long& number);
        friend std::ostream& operator<< (std::ostream &out, const Long& number);
        Long& operator=(const Long& second) {
            low_digits = second.low_digits;
            high_digits = second.high_digits;
            return *this;
        }
        static const unsigned int MAX_UINT = 0-1;
    private :
        unsigned int low_digits;
        unsigned int high_digits;
};
Long operator"" _ull(const char* first);
```

long.cpp:

```
#include "long.h"
Long::Long(std::string number) {
    Long base(0,1), calculate(0, 0);
    const Long ten(0, 10);
    for (int i = number.size() - 1; i >= 0; --i) {
        Long digital(0, number[i]-'0');
        calculate = calculate + digital*base;
        base = base * ten;
    }
    this->high_digits = calculate.high_digits;
    this->low_digits = calculate.low_digits;
}
Long operator"" _ull(const char* first) {
    return Long(std::string(first));
}
Long operator+(const Long& first, const Long& second) {
    Long result;
    result.high_digits = first.high_digits + second.high_digits;
    result.low_digits = first.low_digits + second.low_digits;
    if (result.low_digits < first.low_digits || result.low_digits < second.low_digits) {
// Если при сложении младших разрядов произошло переполнение
        ++result.high_digits; // То увеличиваем значение старших разрядов на 1
    }
    return result;
}
Long operator-(const Long& first, const Long& second) {
    Long result;
    result.low_digits = first.low_digits - second.low_digits;
    result.high_digits = first.high_digits - second.high_digits;
    if (first.low_digits < second.low_digits) { // Если младшие разряды уменьшаемого мень-
ше младших разрядов вычитаемого
        --result.high_digits; // То занимаем 1 у страших разрядов
    }
    return result;
}
Long operator*(const Long& first, const Long& second) {
    Long result;
```

```
    for (int i = 0; i < 32; ++i) { // Умножение младших разрядов обрабатываем как умноже-
ние в "столбик" для переноса полученных значений в старшие разряды
        for (int j = 0; j < 32; ++j) {
            if (((1 << i) & first.low_digits) && ((1 << j) & second.low_digits)) {
                if (i+j >= 32) {
                    result.high_digits += (1 << (i+j-32));
                }
                else {
                    unsigned int old_value = result.low_digits;
                    result.low_digits += (1 << (i+j));
                    if (old_value > result.low_digits)  { // Если при сложении младших
разрядов произошло переполнение
                        ++result.high_digits; // То увеличиваем значение старших разрядов
на 1
                    }
                }
            }
        }
    }
    result.high_digits += first.low_digits * second.high_digits;
    result.high_digits += first.high_digits * second.low_digits;
    return result;
}
bool operator<(const Long& first, const Long& second) {
    if (first.high_digits != second.high_digits) {
        return first.high_digits < second.high_digits;
    }
    return first.low_digits < second.low_digits;
}
bool operator>(const Long& first, const Long& second) {
    return second < first;
}
bool operator==(const Long& first, const Long& second) {
    return (first.high_digits == second.high_digits && first.low_digits == sec-
ond.low_digits);
}
bool operator!=(const Long& first, const Long& second) {
    return !(first == second);
}
Long operator/(const Long& first, const Long& second) {
    Long result;
    assert(second != 0_ull);
    if (second.high_digits == 0) {
        Long check, count, rasn, els, less;
        count.low_digits = Long::MAX_UINT / second.low_digits; // Сколько second содержит-
ся в одном MAX_UINT
        less.low_digits = first.low_digits / second.low_digits; // Деление младших разря-
дов
        els.low_digits = first.high_digits;
        check = count*second*els + less*second;
        rasn = first - check;
        if (second < rasn || rasn == second) { // Если остаток равен или превосходит sec-
ond
            Long plus = rasn / second; // То также делим и его
```

```
            result = result+plus;
        }
        result = result+(count*els) + less;
    }
    else {
        Long check;
        result.low_digits = first.high_digits/second.high_digits; // Деление без учета
младших разрядов
        check = result*second;
        if (first < check) { // Если при делении без учета младших разрядов мы получили
число больше
            Long rasn = check - first; // То необходмо уменьшить частное
            Long minus = rasn / second;
            if (minus * second != rasn) {
                 minus = minus + 1_ull;
            }
            result = result - minus;
        }
        else if (first > check && ((first-check) > second || (first-check)==second)) {
            Long plus = (first-check) / second; // То также делим и его
            result = result+plus;
        }
    }
    return result;
}
Long operator%(const Long& first, const Long& second) {
    assert(second != 0_ull);
    Long result = first - (first / second) * second;
    return result;
}
std::istream& operator>> (std::istream &in, Long& number) {
    std::string input;
    in >> input;
    number = Long(input);
    return in;
}
std::ostream& operator<< (std::ostream &out, const Long& number) {
    Long work = number;
    std::string output;
    if (work == 0_ull) {
        output.insert(0, 1, '0');
    }
    else {
        while (work != 0_ull) {
            Long digit = work % 10_ull;
            output.insert(0, 1, '0'+digit.low_digits);
            work = work / 10_ull;
        }
    }
    out << output;
    return out;
}
```

main.cpp:

```
#include "long.h"
#include <iostream>
int main() {
    std::cout << "Введите выражения в формате: <число1> <оператор> <число2> " <<
std::endl;
    std::cout << "Поддерживаемые операторы: +, -, *, /, %, >, <, ==, != " << std::endl;
    while (true)
    {
        std::string operate;
        Long number1, number2;
        std::cout << "> ";
        std::cin >> number1 >> operate >> number2;
        if (operate == "+") {
            std::cout << "Результат: " << number1 + number2 << std::endl;
        }
        else if (operate == "-") {
            std::cout << "Результат: " << number1 - number2 << std::endl;
        }
        else if (operate == "*") {
            std::cout << "Результат: " << number1 * number2 << std::endl;
        }
        else if (operate == "/") {
            std::cout << "Результат: " << number1 / number2 << std::endl;
        }
        else if (operate == "%") {
            std::cout << "Результат: " << number1 % number2 << std::endl;
        }
        else if (operate == ">") {
            std::cout << "Результат: " << (number1 > number2? "Истина" : "Ложь") <<
std::endl;
        }
        else if (operate == "<") {
            std::cout << "Результат: " << (number1 < number2? "Истина" : "Ложь") <<
std::endl;
        }
        else if (operate == "==") {
            std::cout << "Результат: " << (number1 == number2? "Истина" : "Ложь") <<
std::endl;
        }
        else if (operate == "!=") {
            std::cout << "Результат: " << (number1 != number2? "Истина" : "Ложь") <<
std::endl;
        }
        else {
            std::cout << "Ошибка! Неправильный ввод!" << std::endl;
        }
    }
}
```