**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Перегрузка операторов в С++

Студент: Кочисов Алексей

Группа: М8О-201Б-20

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

**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 С++, и сравнения. Операции необходимо реализовать в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа **Long**.

1. **Описание программы**

Для реализации беззнакового числа из 64 бит опишем класс **UnsignedLL** со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| low\_digits | unsigned int | Хранит младшие разряды, первая часть числа. |
| high\_digits | unsigned int | Хранит старшие разряды, вторая часть числа. |

В классе **UnsignedLL** будут реализованы следующие методы:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| UnsignedLL() | Конструктор по умолчанию, создает число 0. |
| UnsignedLL(std::string number) | Конструктор, создающий беззнаковое число по его строковой записи. |
| UnsignedLL(unsigned int first, unsigned int second) | Конструктор, создающий беззнаковое число по входному значению старших и младших разрядов соответственно. |
| UnsignedLL& operator=(const UnsignedLL& second) | Оператор присваивания для беззнаковых чисел. |

Также для класса **UnsignedLL** будут объявлены как дружественные следующие методы:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| friend UnsignedLL operator+(const UnsignedLL&, const UnsignedLL&) | Операция сложения беззнаковых чисел. |
| friend UnsignedLL operator-(const UnsignedLL&, const UnsignedLL&) | Операция вычитания беззнаковых чисел. |
| friend UnsignedLL operator\*(const UnsignedLL&, const UnsignedLL&) | Операция умножения беззнаковых чисел. |
| friend UnsignedLL operator/(const UnsignedLL&, const UnsignedLL&) | Операция целочисленного деления беззнаковых чисел. |
| friend UnsignedLL operator%(const UnsignedLL&t, const UnsignedLL&) | Операция получения остатка от целочисленного деления беззнаковых чисел. |
| friend bool operator<(const UnsignedLL&, const UnsignedLL&) | Операция сравнения меньше для беззнаковых чисел. |
| friend bool operator==(const UnsignedLL&, const UnsignedLL&) | Операция сравнения равно для беззнаковых чисел. |
| friend bool operator>(const UnsignedLL&t, const UnsignedLL&) | Операция сравнения больше для беззнаковых чисел. |
| friend bool operator!=(const UnsignedLL&, const UnsignedLL&) | Операция сравнения не равно для беззнаковых чисел. |
| friend std::istream& operator>> (std::istream &, UnsignedLL&) | Стандартный оператор ввода для беззнаковых чисел. |
| friend std::ostream& operator<< (std::ostream &, const UnsignedLL& ) | Стандартный оператор вывода для беззнаковых чисел. |
| UnsignedLL operator"" \_ull(const char\* first) | Определенный пользовательский литерал для написания беззнаковых чисел. |

1. **Руководство по использованию программы**

**oop\_exercise\_02** - исполняемый файл программы.

**tests1.txt** - первый набор тестов для программы.

**tests2.txt** - второй набор тестов для программы.

**Входные данные**

Для работы с программой необходимо вводить данные в следующем формате:

<число1> <операция> <число2> , поддерживаемые операции: +, -, \*, /, %, >, <, ==, !=.

**Выходные данные**

Для каждого запроса на выполнение операции выводится ответ в стандартный поток вывода.

**Тестовые примеры**

> 0 - 1

Результат: 18446744073709551615

> 18446744073709551615 \* 100000

Результат: 18446744073709451616

> 18446744073709451616 % 100000

Результат: 51616

> 18446744073709451616 / 1000000009

Результат: 18446743907

> 2000 == 2000

Результат: Истина

> 2 < 200

Результат: Истина

> 12312515 > 1251251242134124

Результат: Ложь

> 1903573895713051 % 3124124

Результат: 2045375

> 1111 != 1111111111111111

Результат: Истина

1. **Листинг программы**

Исходный код программы состоит из двух файлов: **unsignedll.hpp** - содержит объявление и реализацию класса **UnsignedLL**, а также все функции, которые с ним работают; и **main.cpp** - основная программа. **CMakeLists.txt** - содержит код для утилиты cmake, которая проводит сборку программы из исходного кода.

**Содержимое файла unsignedll.hpp**

/\* Кочисов Алексей, группа М8О-201Б-20\*/

#include <iostream>

#include <string>

#include <cassert>

class UnsignedLL {

public :

UnsignedLL() : low\_digits(0), high\_digits(0) {}

UnsignedLL(std::string number);

UnsignedLL(unsigned int first, unsigned int second) : high\_digits(first), low\_digits(second) {}

friend UnsignedLL operator+(const UnsignedLL& first, const UnsignedLL& second);

friend UnsignedLL operator-(const UnsignedLL& first, const UnsignedLL& second);

friend UnsignedLL operator\*(const UnsignedLL& first, const UnsignedLL& second);

friend UnsignedLL operator/(const UnsignedLL& first, const UnsignedLL& second);

friend UnsignedLL operator%(const UnsignedLL& first, const UnsignedLL& second);

friend bool operator<(const UnsignedLL& first, const UnsignedLL& second);

friend bool operator==(const UnsignedLL& first, const UnsignedLL& second);

friend bool operator>(const UnsignedLL& first, const UnsignedLL& second);

friend bool operator!=(const UnsignedLL& first, const UnsignedLL& second);

friend std::istream& operator>> (std::istream &in, UnsignedLL& number);

friend std::ostream& operator<< (std::ostream &out, const UnsignedLL& number);

UnsignedLL& operator=(const UnsignedLL& 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;

};

UnsignedLL::UnsignedLL(std::string number) {

UnsignedLL radix(0,1), calculate(0, 0);

const UnsignedLL ten(0, 10);

for (int i = number.size() - 1; i >= 0; --i) {

UnsignedLL digital(0, number[i]-'0');

calculate = calculate + digital\*radix;

radix = radix \* ten;

}

this->high\_digits = calculate.high\_digits;

this->low\_digits = calculate.low\_digits;

}

UnsignedLL operator"" \_ull(const char\* first) {

return UnsignedLL(std::string(first));

}

UnsignedLL operator+(const UnsignedLL& first, const UnsignedLL& second) {

UnsignedLL 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;

}

UnsignedLL operator-(const UnsignedLL& first, const UnsignedLL& second) {

UnsignedLL 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 меньше младших разрядов вычитаемого

--result.high\_digits; // То занимаем 1 у страших разрядов

}

return result;

}

UnsignedLL operator\*(const UnsignedLL& first, const UnsignedLL& second) {

UnsignedLL result;

for (int i = 0; i < 32; ++i) { // Умножение младших разрядов обрабатываем как умножение 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 UnsignedLL& first, const UnsignedLL& 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 UnsignedLL& first, const UnsignedLL& second) {

return second < first;

}

bool operator==(const UnsignedLL& first, const UnsignedLL& second) {

return (first.high\_digits == second.high\_digits && first.low\_digits == second.low\_digits);

}

bool operator!=(const UnsignedLL& first, const UnsignedLL& second) {

return !(first == second);

}

UnsignedLL operator/(const UnsignedLL& first, const UnsignedLL& second) {

UnsignedLL result;

assert(second != 0\_ull);

if (second.high\_digits == 0) {

UnsignedLL check, count, rasn, els, less;

count.low\_digits = UnsignedLL::MAX\_UINT / second.low\_digits; // Сколько second содержится в одном MAX\_UINT

less.low\_digits = first.low\_digits / second.low\_digits; // Деление младших разрядов

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 second

UnsignedLL plus = rasn / second; // То также делим и его

result = result+plus;

}

result = result+(count\*els) + less;

}

else {

UnsignedLL 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

UnsignedLL rasn = check - first; // То необходмо уменьшить частное

UnsignedLL minus = rasn / second;

if (minus \* second != rasn) {

minus = minus + 1\_ull;

}

result = result - minus;

}

}

return result;

}

UnsignedLL operator%(const UnsignedLL& first, const UnsignedLL& second) {

assert(second != 0\_ull);

UnsignedLL result = first - (first / second) \* second;

return result;

}

std::istream& operator>> (std::istream &in, UnsignedLL& number) {

std::string input;

in >> input;

number = UnsignedLL(input);

return in;

}

std::ostream& operator<< (std::ostream &out, const UnsignedLL& number) {

UnsignedLL work = number;

std::string output;

if (work == 0\_ull) {

output.insert(0, 1, '0');

}

else {

while (work != 0\_ull) {

UnsignedLL digit = work % 10\_ull;

output.insert(0, 1, '0'+digit.low\_digits);

work = work / 10\_ull;

}

}

out << output;

return out;

}

**Содержимое файла main.cpp**

/\* Кочисов Алексей, группа М8О-201Б-20 \*/

#include "unsignedll.hpp"

#include <string>

#include <iostream>

using namespace std;

int main() {

cout << "Введите выражения в формате: <число1> <оператор> <число2> " << endl;

cout << "Поддерживаемые операторы: +, -, \*, /, %, >, <, ==, != " << endl;

while (true)

{

string operate;

UnsignedLL number1, number2;

cout << "> ";

cin >> number1 >> operate >> number2;

if (operate == "+") {

cout << "Результат: " << number1 + number2 << endl;

}

else if (operate == "-") {

cout << "Результат: " << number1 - number2 << endl;

}

else if (operate == "\*") {

cout << "Результат: " << number1 \* number2 << endl;

}

else if (operate == "/") {

cout << "Результат: " << number1 / number2 << endl;

}

else if (operate == "%") {

cout << "Результат: " << number1 % number2 << endl;

}

else if (operate == ">") {

cout << "Результат: " << (number1 > number2? "Истина" : "Ложь") << endl;

}

else if (operate == "<") {

cout << "Результат: " << (number1 < number2? "Истина" : "Ложь") << endl;

}

else if (operate == "==") {

cout << "Результат: " << (number1 == number2? "Истина" : "Ложь") << endl;

}

else if (operate == "!=") {

cout << "Результат: " << (number1 != number2? "Истина" : "Ложь") << endl;

}

else {

cout << "Ошибка! Неправильный ввод!" << endl;

}

}

}

**Содержимое файла CMakeLists.txt**

cmake\_minimum\_required(VERSION 2.8)

set(SOURCES main.cpp)

add\_executable(oop\_exercise\_02 ${SOURCES})

**5. Выводы**

В результате выполнения лабораторной работы я познакомился с возможностями перегрузки операторов при реализации собственного класса в C++, а также научился определять пользовательские литералы для созданных классов.