МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Магнитогорский государственный технический университет

им. Г. И. Носова»

(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Кафедра вычислительной техники и программирования

**Спецификация класса “LongInt”**

по дисциплине «Объектно-ориентированное программное обеспечение»

Исполнитель: Варламов М.Н. студент 2 курса, группа АВб-19-1

Руководитель: Ильина Е.А., доцент каф. ВТиП, к.п.н.

Магнитогорск, 2021

**Описание полей и методов**

string number – Поле для хранения числа.

explicit LongInt(string num) – Конструктор для строковых значений.

explicit LongInt(int num) – Конструктор для целочисленных значений. Конвертирует int в string и записывает значение в number;

explicit LongInt() – Пустой конструктор. Задает полю number значение 0.

LongInt operator+(const LongInt &num) const – Операция суммы. Алгоритм представляет собой складывание чисел в столбик.

LongInt operator-(const LongInt &num) const – Операция разности. Алгоритм представляет собой разность чисел в столбик.

LongInt operator\*(const LongInt &num) const – Операция умножения. Алгоритм представляет собой произведение чисел в столбик.

LongInt operator\*(const int &num) const – Операция умножения для целочисленных значений. Конвертирует int значение в LongInt и выполняет операцию умножения (см. функцию выше).

LongInt operator/(const LongInt &num) const – Операция целочисленного деления. Алгоритм представляет собой деление чисел в столбик.

LongInt operator%(const LongInt &num) const – Операция нахождения остатка от деления. Алгоритм находит остаток от деления по формуле r = n - mk, где n - делимое, m - делитель, k - частное, r – остаток. Вызывает все соответствующие функции.

LongInt &operator=(const LongInt &num) – Оператор присваивания. Присваивает number значение number у параметра num.

bool operator>(const LongInt &num) const – Оператор сравнения “больше”. Сравнивает посимвольно числа, при первом несовпадении возвращает соответствующий результат.

bool operator>=(const LongInt &num) const – Оператор сравнения “больше, либо равно”. Возвращает логическое “Или” между операторами “Больше” и “Равно”.

bool operator<(const LongInt &num) const – Оператор сравнения “меньше”. Работает аналогично оператору “Больше”

bool operator<=(const LongInt &num) const – Оператор сравнения “меньше либо равно”. Работает аналогично оператору “Больше либо равно”

bool operator==(const LongInt &num) const – Оператор сравнения. Сравнивает посимвольно числа, при первом несовпадении возвращает “false”.

bool operator!=(const LongInt &num) const – Оператор сравнения неравенства. Возвращает противоположный результат оператора “Сравнения”.

friend ostream &operator<<(ostream &out, const LongInt &num) – Оператор потокового вывода.

friend const LongInt &operator++(LongInt &num) – Оператор инкрементирования. Вызывает оператор суммы со значением 1.

static LongInt abs(LongInt num) – Операция нахождения модуля.

**Листинг**

**LongInt.h**

#include <string>

#include <malloc.h>

#include <iostream>

#include <sstream>

#include <cstdlib>

#include<bits/stdc++.h>

using namespace std;

class LongInt {

public:

string number;

public :

explicit LongInt(string num);

explicit LongInt(int num);

explicit LongInt();

LongInt operator+(const LongInt &num) const;

LongInt operator-(const LongInt &num) const;

LongInt operator\*(const LongInt &num) const;

LongInt operator\*(const int &num) const;

LongInt operator/(const LongInt &num) const;

LongInt operator%(const LongInt &num) const;

LongInt &operator=(const LongInt &v);

bool operator>(const LongInt &num) const;

bool operator>=(const LongInt &num) const;

bool operator<(const LongInt &num) const;

bool operator<=(const LongInt &num) const;

bool operator==(const LongInt &num) const;

bool operator!=(const LongInt &num) const;

friend ostream &operator<<(ostream &out, const LongInt &num);

friend const LongInt &operator++(LongInt &num);

static LongInt abs(LongInt num);

};

**LongInt.cpp**

#include "LongInt.h”

LongInt::LongInt(string num) {

number = std::move(num);

}

LongInt::LongInt(int num) {

number = to\_string(num);

}

LongInt::LongInt() {

number = "0";

}

LongInt LongInt::operator+(const LongInt &num) const {

if (number.at(0) == '-' && num.number.at(0) != '-')

return num.operator+(\*this); //Проверка

if (number.at(0) != '-' && num.number.at(0) == '-')

return operator-(num \* -1);

stringstream ss;

string max = number, min = num.number;

bool flag = false;

if (max.at(0) == '-' && min.at(0) == '-') {

max.erase(0, 1); //erase(позиция, число символов)

min.erase(0, 1);

flag = true;

}

if (LongInt::abs(\*this) < LongInt::abs(num)) {

string tmp = max;

max = min;

min = tmp;

}

int ost = 0;

int maxInd = max.length() - 1, minInd = min.length() - 1;

for (int i = 0; i < max.length(); i++) {

int a = static\_cast<int>(max[maxInd]) - 48;

if (minInd != -1) {

int b = static\_cast<int>(min[minInd]) - 48;

int c = a + b + ost;

ost = 0;

if (c >= 10) {

c -= 10;

ost++;

}

ss << c;

minInd--;

} else {

int c = a + ost;

ost = 0;

if (c >= 10) {

c -= 10;

ost++;

}

ss << c;

}

maxInd--;

}

if (ost == 1)

ss << 1;

string ans = ss.str();

reverse(ans.begin(), ans.end());

int k = 0;

for (char an : ans) {

if (an != '0')

break;

else

k++;

}

ans.erase(0, k);

if (ans.length() == 0){

ans.append(1, '0');

flag = false;

}

if (flag)

ans.insert(0, "-");

return LongInt(ans);

}

LongInt LongInt::operator-(const LongInt &num) const {

if (number.at(0) == '-' && num.number.at(0) == '-')

return operator+(num \* -1);

if (number.at(0) == '-' && num.number.at(0) != '-')

return operator+(num \* -1);

if (number.at(0) != '-' && num.number.at(0) == '-')

return operator+(num \* -1);

stringstream ss;

string max = number, min = num.number;

bool flag = false;

if (max.at(0) == '-')

max.erase(0, 1);

if (min.at(0) == '-')

min.erase(0, 1);

if (LongInt::abs(\*this) < LongInt::abs(num)) {

string tmp = max;

max = num.number;

min = tmp;

flag = true;

}

int maxInd = max.length() - 1, minInd = min.length() - 1;

for (int i = 0; i < max.length(); i++) {

int a = static\_cast<int>(max[maxInd]) - 48;

if (minInd != -1) {

int b = static\_cast<int>(min[minInd]) - 48;

int c;

if (a - b < 0) {

for (int j = maxInd - 1; j >= 0; j--) {

if (max[j] == '0')

max[j] = (static\_cast<int>(9)) + '0';

else {

max[j] = (static\_cast<int>(max[j]) - 48 - 1) + '0'; //Вот так

break;

}

}

c = (a + 10) - b;

} else

c = a - b;

ss << c;

minInd--;

} else {

ss << a;

}

maxInd--;

}

string ans = ss.str();

reverse(ans.begin(), ans.end());

int k = 0;

for (char an : ans) {

if (an != '0')

break;

else

k++;

}

ans.erase(0, k);

if (ans.length() == 0){

ans.append(1, '0');

flag = false;

}

if (flag)

ans.insert(0, "-");

return LongInt(ans);

}

LongInt LongInt::operator\*(const LongInt &num) const {

if (num == LongInt(0) || \*this == LongInt(0))

return LongInt(0);

string n1 = number, n2 = num.number;

bool flag = false;

if (n1[0] == '-') {

flag = !flag;

n1.erase(0, 1);

}

if (n2[0] == '-') {

flag = !flag;

n2.erase(0, 1);

}

if (n1.length() > n2.length()) // делаем n2 большим по длинне

n1.swap(n2);

string res = "0";

for (int i = n1.length() - 1; i >= 0; --i) // обратный проход по меньшему

{

string temp = n2; // временное = большему

int currentDigit = n1[i] - '0'; // текуший разряд меньшего

int carry = 0;

for (int j = temp.length() - 1; j >= 0; --j) // обратный проход по временному

{

temp[j] = ((temp[j] - '0') \* currentDigit) + carry; // произведение текущих символов плюс остаток

if (temp[j] > 9) // вычисление нового остатка

{

carry = (temp[j] / 10);

temp[j] -= (carry \* 10);

} else

carry = 0;

temp[j] += '0'; // возвращение к цифрам

}

if (carry > 0) // выделение места под новый разряд

temp.insert(0, 1, (carry + '0'));

temp.append((n1.length() - i - 1), '0'); // смещение на разряд

res = (LongInt(res) + LongInt(temp)).number; // суммирование всех поразрядных произведений

}

while (res[0] == '0' && res.length() != 1) // удаление незначащих нулей

res.erase(0, 1);

if (flag)

return LongInt(string("-").append(res));

return LongInt(res);

}

LongInt LongInt::operator\*(const int &num) const {

return operator\*(LongInt(num));

}

LongInt LongInt::operator/(const LongInt &num) const {

if (\*this == (LongInt(0)) && num != LongInt(0))

return LongInt(0);

if (num == LongInt(0))

cerr << "Division by zero";

int flag = 1;

LongInt tmp1 = \*this;

if (\*this < LongInt(0)) {

tmp1 = LongInt::abs(tmp1);

flag \*= -1;

}

LongInt tmp2 = num;

if (tmp2 < LongInt(0)) {

tmp2 = LongInt::abs(tmp2);

flag \*= -1;

}

if (tmp1 < tmp2)

return LongInt(0);

LongInt res(1);

while (tmp1 >= (res \* tmp2)) {

++res;

}

return (res - LongInt(1)) \* flag;

}

LongInt LongInt::operator%(const LongInt &num) const {

if (\*this == (LongInt(0)) && num != LongInt())

return LongInt(0);

if (num == LongInt(0))

cerr << "Division by zero";

if (LongInt::abs((const LongInt &) \*this) < LongInt::abs(num))

return \*this;

return ( \*this - (num \* (\*this/num)));

}

bool LongInt::operator>(const LongInt &num) const {

if (number[0] == '-' && num.number[0] != '-')

return false;

if (number[0] != '-' && num.number[0] == '-')

return true;

string n1 = number, n2 = num.number;

bool flag = true;

if (number[0] == '-' && num.number[0] == '-') {

flag = false;

n1.erase(0, 1);

n2.erase(0, 1);

}

if (n1.length() > n2.length())

return flag;

if (n1.length() < n2.length())

return !flag;

for (int i = 0; i < n1.length(); i++) {

if (static\_cast<int>(n1[i]) > static\_cast<int>(n2[i]))

return (flag);

else if (static\_cast<int>(n1[i]) < static\_cast<int>(n2[i]))

return (!flag);

}

return false;

}

bool LongInt::operator==(const LongInt &num) const {

if (number.length() > num.number.length() || number.length() < num.number.length())

return false;

for (int i = 0; i < number.length(); i++) {

if (static\_cast<int>(number[i]) != static\_cast<int>(num.number[i]))

return false;

}

return true;

}

bool LongInt::operator>=(const LongInt &num) const {

return operator>(num) || operator==(num);

}

bool LongInt::operator<(const LongInt &num) const {

if (number[0] == '-' && num.number[0] != '-')

return true;

if (number[0] != '-' && num.number[0] == '-')

return false;

string n1 = number, n2 = num.number;

bool flag = false;

if (number[0] == '-' && num.number[0] == '-') {

flag = true;

n1.erase(0, 1);

n2.erase(0, 1);

}

if (n1.length() > n2.length())

return flag;

if (n1.length() < n2.length())

return !flag;

for (int i = 0; i < n1.length(); i++) {

if (static\_cast<int>(n1[i]) > static\_cast<int>(n2[i]))

return (flag);

else if (static\_cast<int>(n1[i]) < static\_cast<int>(n2[i]))

return (!flag);

}

return false;

}

bool LongInt::operator<=(const LongInt &num) const {

return operator<(num) || operator==(num);

}

bool LongInt::operator!=(const LongInt &num) const {

return !operator==(num);

}

ostream &operator<<(ostream &out, const LongInt &number) {

out << number.number;

return out;

}

const LongInt &operator++(LongInt &i) {

i = i + (LongInt(1));

return i;

}

LongInt &LongInt::operator=(const LongInt &i) {

if (this == &i) {

return \*this;

}

number.assign(i.number);

return \*this;

}

LongInt LongInt::abs(LongInt num) {

if (num.number[0] == '-')

return LongInt(num.number.erase(0,1));

return num;

}