ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А. В. Устинов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| ОПЕРАТОРНЫЕ ФУНКЦИИ В ПРОСТРАНСТВЕ ИМЕН И КАК ЧЛЕНЫ КЛАССА |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | Д.В.Килин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**1. Цель работы**

Изучить принципы построения приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Закрепить знания по теме: Операторные функции в пространстве имен и как члены класса.

****

**2. Листинг программы**

#pragma once

#include <vector>

#include <string>

#include <memory>

**BIG.H**

class big {

public:

big(std::vector<int> &array1, bool&znk);

big(std::string &input);

void operator=(big &input);

bool operator >(big &input);

big operator -(big &input);

void NormalLenght(std::vector<int> &array1, std::vector<int> &array2);

void set\_num(std::string &input);

void get\_num();

big summ(big &num1, big &num2);

big diff(big &num1, big &num2, bool &PlaseOfCall);

private:

void FindBiggest(std::vector<int> &array1, std::vector<int> &array2, int &big);

std::vector<int> VecSumm(std::vector<int> &array1, std::vector<int> &array2);

void Multiplication(std::vector<int> big, std::vector<int> small, std::vector<int>& result);

std::vector<int> mVector;

std::shared\_ptr<std::vector<int>> array = std::make\_shared<std::vector<int> >(std::move(mVector));

std::shared\_ptr<bool> znak;

};

**BIG.CPP**

#include "Big.h"

#include <string>

#include<iostream>

#include <vector>

#include <memory>

using namespace std;

big::big(string &input) {

set\_num(input);

}

big::big(vector<int> &array1,bool &znk) {

for (int i = 0; i < array1.size(); i++) {

(\*array).insert((\*array).end(), array1[i]);

}

znak = std::make\_shared<bool>(znk);

}

void big::set\_num(std::string &input)

{

bool znk = 0;

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

znk = 1;

input.erase(0, 1);

}

if (input[0] == '+') {

znk = 0;

input.erase(0, 1);

}

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

(\*array).insert((\*array).end(), input[i] - 48);

}

znak = std::make\_shared<bool>(znk);

}

void big::get\_num() {

if (\*znak == 0) cout << "+";

else cout << "-";

for (int i = 0; i < (\*array).size(); i++) {

cout << (\*array)[i];

}

}

void big::NormalLenght(vector<int> &array1, vector<int> &array2) {

int raz = (array1.size() - array2.size());

if (array1.size() > array2.size()) {

for (int i = 0; i <raz; i++) {

array2.insert(array2.begin(), 0);

}

}

else if (array1.size() < array2.size()) {

for (int i = 0; i < (raz\*(-1)); i++) {

array1.insert(array1.begin(), 0);

}

}

}

void big::FindBiggest(std::vector<int> &array1, std::vector<int> &array2,int &biggest) {

for (int i = 0; i < array1.size(); i++) {

if (array1[i] > array2[i]) {

biggest = 1;

break;

}

else if (array1[i] < array2[i]) {

biggest = 2;

break;

}

}

}

void big::operator=(big &input) {

(\*array).clear();

for (int i = 0; i < (\*input.array).size(); i++) {

(\*array).insert((\*array).end(), (\*input.array)[i]);

}

znak = std::make\_shared<bool>(\*input.znak);

}

bool big::operator >(big &input) {

if ((\*znak == 0) && ((\*input.znak) == 1)) {

return 1;

}

else if ((\*znak == 1) && ((\*input.znak) == 0)) {

return 0;

}

else {

int biggest;

NormalLenght((\*array), (\*input.array));

FindBiggest((\*array), (\*input.array), biggest);

if (((\*znak == 0) && ((\*input.znak) == 0))) {

if (biggest == 1) return 1;

else return 0;

}

else {

if (biggest == 1) return 0;

else return 1;

}

}

}

big big::operator -(big &input) {

int biggest = 0;

int buff = 0;

int curr;

vector<int> result;

if (\*input.znak == 1) {

\*input.znak = 0;

big out = input.summ(\*this, input);

return out;

}

NormalLenght((\*array), (\*input.array));

FindBiggest((\*array), (\*input.array), biggest);

if (biggest == 1) {

for (int i = ((\*array)).size() - 1; i > -1; i--) {

curr = ((\*array)[i] - buff) - (\*input.array)[i];

if (curr < 0) {

curr = ((10 + (\*array)[i] - buff) - (\*input.array)[i]);

buff = 1;

}

else if (curr >= 0) {

buff = 0;

}

result.insert(result.begin(), curr);

}

// cout << endl << "+" << endl;

while (result[0] == 0) {

result.erase(result.begin() + 0);

}

big out(result, \*znak);

return out;

}

if (biggest == 2) {

for (int i = (\*array).size() - 1; i > -1; i--) {

curr = ((\*input.array)[i] - buff) - (\*array)[i];

if (curr < 0) {

curr = ((10 + (\*input.array)[i]) - (\*input.array)[i]);

buff = 1;

}

else if (curr >= 0) {

buff = 0;

}

result.insert(result.begin(), curr);

}

while (result[0] == 0) {

result.erase(result.begin() + 0);

}

big out(result, \*input.znak);

return out;

}

if (biggest == 0) {

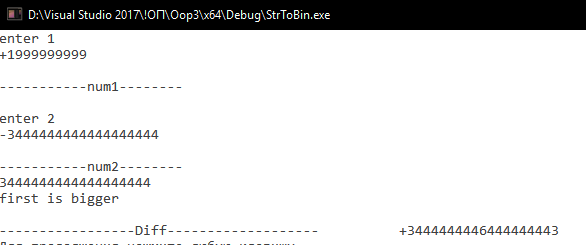
result.insert(result.begin(), 0);

big out(result, \*input.znak);

}

}

**3. Пример работы**



**4. Вывод**

Изучил принципы построения приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Добавил в программу несколько перегруженных операторов, согласно номеру варианта.