ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

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

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
| СТАНДАРТНАЯ БИБЛИОТЕКА С++. БИБЛИОТЕКА ВВОДА-ВЫВОДА |
| 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 консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

**2. Листинг программы**

#pragma once

#include <vector>

#include <string>

#include <memory>

class big {

public:

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

big(std::string &input);

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

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

bool get\_num();

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

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

big Prepair\_for\_Mult(big &num1, big &num2);

big Division(big &num1, big &num2, bool &CallAsMod);

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;

};

#include<iostream>

#include<string>

#include <vector>

#include<cstdint>

#include <algorithm>

#include<cmath>

#include "Big.h"

#define \_CRTDBG\_MAP\_ALLOC

#include <crtdbg.h>

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new ( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define newDBG\_NEW

#endif

#endif

using namespace std;

void SubMenu(big num1);

string check() {

string str;

int err = 1;

int count;

//char a = '1';//49

//char b = '9';//57

//cout << (int)a << endl;

//cout << (int)b;

//|| ((int)(str[0]>48)) || ((int)(str[0]<56)

cin >> str;

//cout << (int)str[0] << endl;

while (!((str[0]=='+')|| (str[0] == '-')|| ((int)(str[0]>48)) || ((int)(str[0]<56))))

{

cout << "incorrect expression\n"

<< "try again\n";

cin >> str;

}

while (err!=0) {

err = 0;

for (int i = 1; i < str.length(); i++) {

if ((((int)str[i]) < 48) || (((int)str[i]) > 57)) {

err++;

cout << "incorrect expression\n"

<< "try again\n";

str = check();

}

}

}

return str;

}

void Newnums() {

cout << endl << "Enter nums"<<endl;

string input;

cout << "enter 1" << endl;

input = check();

big num1(input);

input.clear();

cout << endl << "enter 2" << endl;

cin >> input;

big num2(input);

cout << endl;

int Choise;

cout << endl << "Choose Action\n"

<< "(1)Summ\n"

<< "(2)Diff\n"

<< "(3)Mult\n"

<< "(4)Div\n"

<< "(5)Mod\n"

<< "(6)EXIT\n";

cin >> Choise;

switch (Choise)

{

case 1: {

big out = num1.summ(num1, num2);

out.get\_num();

SubMenu(out);

break;

}

case 2: {

bool PlaseOfCall = 1;

big out = num1.diff(num1, num2, PlaseOfCall);

out.get\_num();

SubMenu(out);

break;

}

case 3: {

big out = num1.Prepair\_for\_Mult(num1, num2);

out.get\_num();

SubMenu(out);

break;

}

case 4: {

bool CallAsMod = 0;

big out = num1.Division(num1, num2, CallAsMod);

bool ErrFlag = out.get\_num();

if (ErrFlag) {

int Choise1;

cout << endl << "Choose Action\n"

<< "(1)Continue\n"

<< "(2)Exit\n";

cin >> Choise1;

switch (Choise1)

{

case 1: {

Newnums();

break;}

case 2: {

exit(0);

}

SubMenu(out);

break;

default: {

cout << "incorrect expression\n"

<< "try again\n";

}

}

}

else {

SubMenu(out);

}

}

case 5: {

bool CallAsMod = 1;

big out = num1.Division(num1, num2, CallAsMod);

out.get\_num();

SubMenu(out);

break;

}

case 6: {

exit(0);

}

default: {

cout << "incorrect expression\n"

<< "try again\n";

}

}

}

void SubMenu(big num1) {

int Choise;

cout << endl << "Next action?\n"

<< "(1)Continue with this num\n"

<< "(2)Enter new nums\n"

<< "(3)EXIT\n";

cin >> Choise;

switch (Choise)

{

case 1: {

Continue(num1);

break;

}

case 2: {

Newnums();

break;

}

case 3: {

exit(0);

}

default: {

cout << "incorrect expression\n"

<< "try again\n";

}

}

}

int main() {

Newnums();

// Для обнаружения утечек памяти

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

system("pause");

return 0;

}

#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);

}

bool big::get\_num() {

if (((\*array)[0] == 0) && (\*znak == 1)) {

return 1;

}

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

else cout << "-";

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

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

}

return 0;

}

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;

}

}

}

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

vector<int> result;

NormalLenght(array1,array2);

int curr;

int buff=0;

for (int i = array1.size()-1; i > -1;i--) {

curr = (array1[i] + array2[i])+buff;

if (curr < 10) {

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

buff = 0;

}

else {

result.insert(result.begin(), (curr%10));

buff = curr/10;

}

}

if (buff != 0) {

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

}

return result;

}

big big::summ(big &num1,big &num2) {

bool PlaceOfCall = 0;

//if both numbers are positive or negative

if ((\*num1.znak == 0) && (\*num2.znak == 0) || (\*num1.znak == 1) && (\*num2.znak == 1)) {

vector<int> result = VecSumm((\*num1.array), (\*num2.array));

big out(result, \*num2.znak);

return out;

}

else {

big out = num1.diff(num1, num2,PlaceOfCall);

return out;

}

}

big big::diff(big &num1, big &num2, bool &PlaseOfCall) {

int biggest = 0;

int buff = 0;

int curr;

vector<int> result;

if ((PlaseOfCall == 1) && (\*num2.znak == 1)) {

\*num2.znak = 0;

big out = num1.summ(num1, num2);

return out;

}

else {

NormalLenght((\*num1.array), (\*num2.array));

FindBiggest((\*num1.array), (\*num2.array), biggest);

if (biggest == 1) {

for (int i = (\*num1.array).size() - 1; i > -1; i--) {

curr = ((\*num1.array)[i] - buff) - (\*num2.array)[i];

if (curr < 0) {

curr = ((10 + (\*num1.array)[i] - buff) - (\*num2.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, \*num1.znak);

return out;

}

if (biggest == 2) {

for (int i = (\*num1.array).size() - 1; i > -1; i--) {

curr = ((\*num2.array)[i] - buff) - (\*num1.array)[i];

if (curr < 0) {

curr = ((10 + (\*num2.array)[i]) - (\*num1.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, \*num2.znak);

return out;

}

if (biggest == 0) {

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

big out(result, \*num2.znak);

}

}

}

big big::Prepair\_for\_Mult(big &num1, big &num2) {

vector<int> result;

if (((\*num1.array)[0] == 0) || ((\*num2.array)[0] == 0)) {

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

bool znk = 0;

big out(result, znk);

return out;

}

int biggest;

NormalLenght((\*num1.array), (\*num2.array));

FindBiggest((\*num1.array), (\*num2.array), biggest);

if (biggest == 1){

while ((\*num2.array)[0] == 0) {

(\*num2.array).erase((\*num2.array).begin() + 0);

}

Multiplication((\*num1.array), (\*num2.array), result);

}

else

{

while ((\*num1.array)[0] == 0)

{

(\*num1.array).erase((\*num1.array).begin() + 0);

}

Multiplication((\*num2.array), (\*num1.array), result);

}

//TODO think about return znak CLOSED

bool znk;

if ((((\*num1.znak)==0)&&((\*num2.znak)==0))|| (((\*num1.znak) == 1) && ((\*num2.znak) == 1))) {

znk = 0;

}

else {

znk = 1;

}

big out(result, znk);

return out;

}

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

vector<int> currentsumm;

vector<int> temp;

int nul = 0;

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

if (small[i] == 0) {

continue;

}

else if (small[i] == 1)

{

temp = big;

}

else {

temp = big;

for (int j = 0; j < (small[i] - 1); j++) {

currentsumm=VecSumm(temp, big);

temp = currentsumm;

currentsumm.clear();

}

}

for (int d = small.size() - 1; d > i; d--) {

temp.insert(temp.end(), 0);

}

currentsumm = result;

result.clear();

result=VecSumm(temp, currentsumm);

temp.clear();

currentsumm.clear();

}

}

big big::Division(big &num1, big &num2,bool &CallAsMod) {

vector<int> answer;

bool PlaceOfCall = 0;

vector<int> one;

if ((\*num2.array)[0] == 0){

cout << "division error" << endl;

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

bool znk = 1;

big out(answer, znk);

return out;

}

int biggest = 0;

bool znk = 0;

one.insert(one.end(), 1);

big ones(one, znk);

answer.insert(answer.end(), 1);

big ans(answer, znk);

big result(answer, znk);

if ((\*num2.znak) == 1) znk = 1;

if ((CallAsMod == 1) && ((\*num2.array)[0]) == 1) {

vector<int> nul;

nul.insert(nul.end(), 0);

big null(nul, znk);

return null;

}

while (biggest != 2) {

result =Prepair\_for\_Mult(num2, ans);

NormalLenght(\*num1.array, \*result.array);

for (int i = 0; i < (\*num1.array).size(); i++) {

if ((\*num1.array)[i] > (\*result.array)[i]) {

biggest = 1;

break;

}

else if ((\*num1.array)[i] < (\*result.array)[i]) {

biggest = 2;

break;

}

}

if (biggest == 2) break;

(\*result.array).clear();

(\*result.array)=VecSumm((\*ans.array), one);

(\*ans.array) = (\*result.array);

(\*result.array).clear();

}

if (CallAsMod == 0) {

result = diff(ans, ones, PlaceOfCall);

if ((((\*num1.znak) == 0) && ((\*num2.znak) == 0)) || (((\*num1.znak) == 1) && ((\*num2.znak) == 1)))

{

\*result.znak = 0;

}

else \*result.znak = 1; {

if ((\*result.array)[0] == 0) {

\*result.znak = 1;

}

}

return result;

}

// MOD

else {

result = diff(ans, ones, PlaceOfCall);

ans = Prepair\_for\_Mult(num2, result);

result = diff(num1, ans, PlaceOfCall);

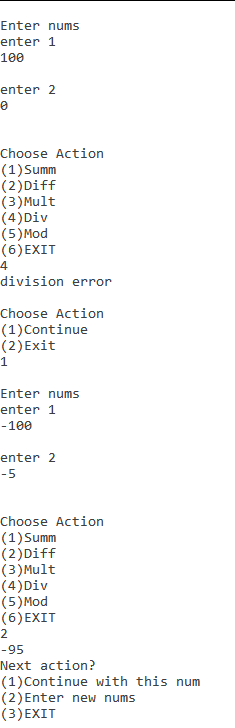
\*result.znak = znk;

return result;

}

}

**3. Пример работы**



**4. Вывод**

Изучил принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.  
Научился получать и выводить информацию о типе переменной в C++. Дополнил варианты взаимодействия с программой для вычисления больших чисел через консоль.