Queno. 1

→

**#ifndef COMPLEX\_H**

**#define COMPLEX\_H**

**#include <iostream>**

**class Complex {**

**public:**

**explicit Complex(double = 0.0, double = 0.0); // constructor**

**Complex operator+(const Complex &) const; // addition**

**Complex operator-(const Complex &) const; // subtraction**

**Complex operator\*(const Complex &) const; // multiplication**

**bool operator==(const Complex &) const; // equality comparison**

**bool operator!=(const Complex &) const; // inequality comparison**

**friend std::ostream &operator<<(std::ostream &, const Complex &);**

**friend std::istream &operator>>(std::istream &, Complex &);**

**private:**

**double real; // real part**

**double imaginary; // imaginary part**

**};**

**#endif**

**#include "complex.h"**

**using namespace std;**

**Complex::Complex(double realPart, double imaginaryPart)**

**: real(realPart), imaginary(imaginaryPart) {}**

**Complex Complex::operator+(const Complex &operand2) const {**

**return Complex(real + operand2.real, imaginary + operand2.imaginary);**

**}**

**Complex Complex::operator-(const Complex &operand2) const {**

**return Complex(real - operand2.real, imaginary - operand2.imaginary);**

**}**

**Complex Complex::operator\*(const Complex &operand2) const {**

**return Complex(real \* operand2.real - imaginary \* operand2.imaginary,**

**real \* operand2.imaginary + imaginary \* operand2.real);**

**}**

**bool Complex::operator==(const Complex &operand2) const {**

**return real == operand2.real && imaginary == operand2.imaginary;**

**}**

**bool Complex::operator!=(const Complex &operand2) const {**

**return !(\*this == operand2);**

**}**

**ostream &operator<<(ostream &out, const Complex &c) {**

**out << '(' << c.real << ", " << c.imaginary << ')';**

**return out;**

**}**

**istream &operator>>(istream &in, Complex &c) {**

**in >> c.real >> c.imaginary;**

**return in;**

**}**

**#include "complex.h"**

**#include <iostream>**

**using namespace std;**

**int main(void) {**

**Complex x;**

**Complex y(4.3, 8.2);**

**Complex z(3.3, 1.1);**

**cout << "x: " << x << "\ny: " << y << "\nz: " << z;**

**x = y + z;**

**cout << "\n\nx = y + z:\n" << x << " = " << y << " + " << z;**

**x = y - z;**

**cout << "\n\nx = y - z:\n" << x << " = " << y << " - " << z;**

**x = y \* z;**

**cout << "\n\nx = y \* z:\n" << x << " = " << y << " \* " << z;**

**if (x == y) {**

**cout << "\n\nx is equal to y\n";**

**} else {**

**cout << "\n\nx is not equal to y\n";**

**}**

**cout << endl;**

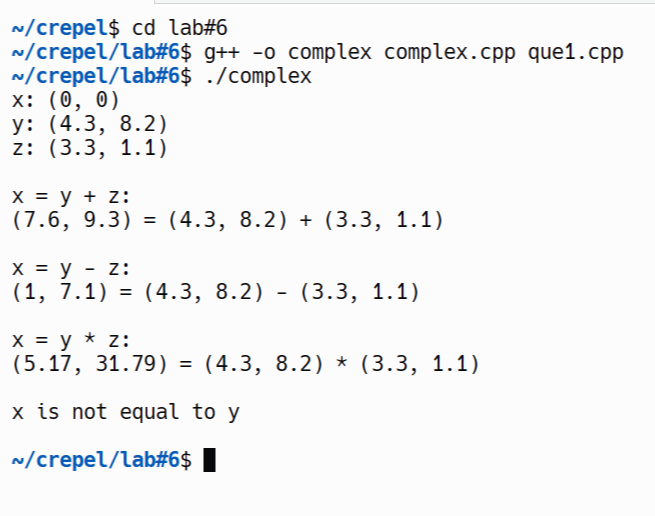
**return 0;**

**}**

Files created:



Output:



**Queno. 2**

**// Hugeint.h**

**#ifndef HUGEINT\_H**

**#define HUGEINT\_H**

**#include <array>**

**#include <iostream>**

**#include <string>**

**class HugeInt {**

**friend std::ostream &operator<<(std::ostream &, const HugeInt &);**

**public:**

**static const int digits = 30; // maximum digits in a HugeInt**

**HugeInt(long = 0); // conversion/default constructor**

**HugeInt(const std::string &); // conversion constructor**

**HugeInt operator+(const HugeInt &) const;**

**HugeInt operator+(int) const;**

**HugeInt operator+(const std::string &) const;**

**HugeInt operator\*(const HugeInt &) const;**

**HugeInt operator/(const HugeInt &) const;**

**bool operator==(const HugeInt &) const;**

**bool operator!=(const HugeInt &) const;**

**bool operator<(const HugeInt &) const;**

**bool operator<=(const HugeInt &) const;**

**bool operator>(const HugeInt &) const;**

**bool operator>=(const HugeInt &) const;**

**private:**

**std::array<short, digits> integer;**

**};**

**#endif**

**// Hugeint.cpp**

**#include "Hugeint.h"**

**using namespace std;**

**HugeInt::HugeInt(long value) {**

**for (short &element : integer)**

**element = 0;**

**for (size\_t j = digits - 1; value != 0 && j >= 0; j--) {**

**integer[j] = value % 10;**

**value /= 10;**

**}**

**}**

**HugeInt::HugeInt(const string &number) {**

**for (short &element : integer)**

**element = 0;**

**size\_t length = number.size();**

**for (size\_t j = digits - length, k = 0; j < digits; ++j, ++k)**

**if (isdigit(number[k]))**

**integer[j] = number[k] - '0';**

**}**

**HugeInt HugeInt::operator+(const HugeInt &op2) const {**

**HugeInt temp;**

**int carry = 0;**

**for (int i = digits - 1; i >= 0; i--) {**

**temp.integer[i] = integer[i] + op2.integer[i] + carry;**

**if (temp.integer[i] > 9) {**

**temp.integer[i] %= 10;**

**carry = 1;**

**} else**

**carry = 0;**

**}**

**return temp;**

**}**

**HugeInt HugeInt::operator+(int op2) const {**

**return \*this + HugeInt(op2);**

**}**

**HugeInt HugeInt::operator+(const string &op2) const {**

**return \*this + HugeInt(op2);**

**}**

**HugeInt HugeInt::operator\*(const HugeInt &op2) const {**

**HugeInt result;**

**for (int i = digits - 1; i >= 0; i--) {**

**int carry = 0;**

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

**int product = result.integer[i + j + 1] + integer[j] \* op2.integer[i] + carry;**

**result.integer[i + j + 1] = product % 10;**

**carry = product / 10;**

**}**

**result.integer[i] += carry;**

**}**

**return result;**

**}**

**// Division operation is complex and not implemented here**

**HugeInt HugeInt::operator/(const HugeInt &op2) const {**

**return HugeInt(0);**

**}**

**bool HugeInt::operator==(const HugeInt &op2) const {**

**return integer == op2.integer;**

**}**

**bool HugeInt::operator!=(const HugeInt &op2) const {**

**return !(\*this == op2);**

**}**

**bool HugeInt::operator<(const HugeInt &op2) const {**

**for (int i = 0; i < digits; i++) {**

**if (integer[i] < op2.integer[i]) return true;**

**if (integer[i] > op2.integer[i]) return false;**

**}**

**return false; // equal**

**}**

**bool HugeInt::operator<=(const HugeInt &op2) const {**

**return \*this < op2 || \*this == op2;**

**}**

**bool HugeInt::operator>(const HugeInt &op2) const {**

**return !(\*this <= op2);**

**}**

**bool HugeInt::operator>=(const HugeInt &op2) const {**

**return \*this > op2 || \*this == op2;**

**}**

**ostream &operator<<(ostream &output, const HugeInt &num) {**

**int i;**

**for (i = 0; (i < HugeInt::digits) && (0 == num.integer[i]); ++i);**

**if (i == HugeInt::digits)**

**output << 0;**

**else**

**for (; i < HugeInt::digits; ++i)**

**output << num.integer[i];**

**return output;**

**}**

**// main.cpp**

**#include <iostream>**

**#include "Hugeint.h"**

**using namespace std;**

**int main(void) {**

**HugeInt n1(7654321);**

**HugeInt n2(7891234);**

**HugeInt n3("99999999999999999999999999999");**

**HugeInt n4("1");**

**HugeInt n5;**

**cout << "n1 is " << n1 << "\nn2 is " << n2**

**<< "\nn3 is " << n3 << "\nn4 is " << n4**

**<< "\nn5 is " << n5 << "\n\n";**

**n5 = n1 + n2;**

**cout << n1 << " + " << n2 << " = " << n5 << "\n\n";**

**cout << n3 << " + " << n4 << "\n= " << (n3 + n4) << "\n\n";**

**n5 = n1 + 9;**

**cout << n1 << " + " << 9 << " = " << n5 << "\n\n";**

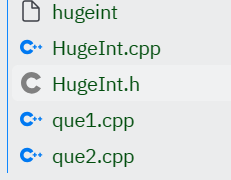
**n5 = n2 + "10000";**

**cout << n2 << " + " << "10000" << " = " << n5 << endl;**

**return 0;**

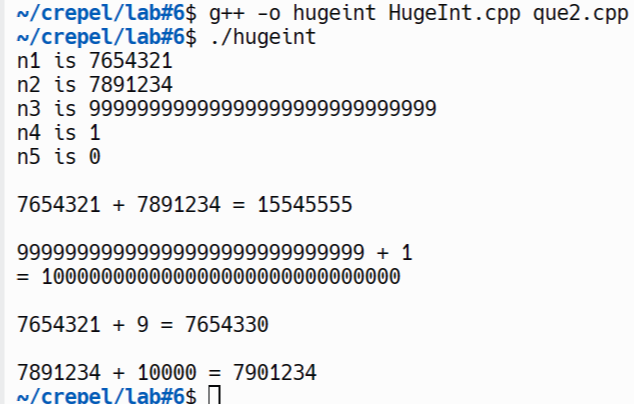
**}**

**Files:**

****

**output**

**;**

****