**Лабораторна робота №3**

**Тема:** Перевантаження операцій класу

**Мета:** ознайомитись зі способами перевантаження операцій та навчитись використовувати їх при роботі з об’єктами

**Завдання 1.** В класі Int, який розроблений в завданні №1 лабораторної роботи №1, перевизначити чотири цілочисельні арифметичні операції («+», «-», «\*» , «/») так, щоб їх можна було використовувати для операцій з об'єктами класу Int.

Якщо результат будь-якої з операцій виходить за межі типу int (в 32-бітній системі), що може мати значення від 2 147 483 648 до -2 147 483 648, то операція повинна послати повідомлення про помилку і завершити програму. Такі типи даних корисні там, де помилки можуть бути викликані арифметичним переповненням, яке неприпустимо. Напишіть програму для перевірки цього класу. Підказка: для полегшення перевірки переповнення виконуйте обчислення з використанням типу long double.

**Код програми:**

#include <QCoreApplication>

#include <iostream>

#include <limits.h>

class **IntA**{

private:

long int m\_a;

public:

**IntA**() {m\_a = 0;}

**IntA**(int a){ m\_a = a; }

void **setNum**(){

std::cout << "Enter the number a";

std::cin >> m\_a; }

void **toNull**(){ m\_a = 0; }

void **showA**() const{

std::cout << "a is "<< m\_a; }

int **getA**(){ return m\_a; }

void **sum**(IntA b, IntA c){

m\_a = b.getA() + c.getA(); }

friend IntA operator+(const IntA &a, const IntA &b);

friend IntA operator-(const IntA &a, const IntA &b);

friend IntA operator/(const IntA &a, const IntA &b);

friend IntA operator\*(const IntA &a, const IntA &b); };

IntA operator+(const IntA &a, const IntA &b)

{

if ((b.m\_a > 0) && (a.m\_a > INT\_MAX - b.m\_a))

{ std::cout << "error"; exit(0);}

if ((b.m\_a < 0) && (a.m\_a < INT\_MIN - b.m\_a))

{ std::cout << "error"; exit(0);};

return(a.m\_a+b.m\_a);

}

IntA operator-(const IntA &a, const IntA &b)

{

if ((b.m\_a < 0) && (a.m\_a > INT\_MAX + b.m\_a))

{ std::cout << "error"; exit(0);}

if ((b.m\_a > 0) && (a.m\_a < INT\_MIN + b.m\_a))

{ std::cout << "error"; exit(0);}

return(a.m\_a-b.m\_a); }

IntA operator/(const IntA &a, const IntA &b)

{

return(a.m\_a/b.m\_a);

}

IntA operator\*(const IntA &a, const IntA &b)

{ if (a.m\_a > INT\_MAX / b.m\_a)

{ std::cout << "error"; exit(0);}

if ((a.m\_a < INT\_MIN / b.m\_a))

{ std::cout << "error"; exit(0);}   
 if ((a.m\_a == -1) && (b.m\_a == INT\_MIN))

{ std::cout << "error"; exit(0);}

if ((a.m\_a == -1) && (b.m\_a == INT\_MIN))

{ std::cout << "error"; exit(0);}

return(a.m\_a\*b.m\_a); }

int **main**(int argc, char \*argv[]) {

QCoreApplication a(*argc*, argv);

IntA c(5),b(8),n3,n4(2147483641);

std::cout << "b is" << b.getA() << std::endl;

std::cout << "c is" << c.getA() << std::endl;

n3=c+b;

std::cout << "c+b " << n3.getA() << std::endl;

n3=c-b;

std::cout << "c-b = " << n3.getA() << std::endl;

n3=c/b;

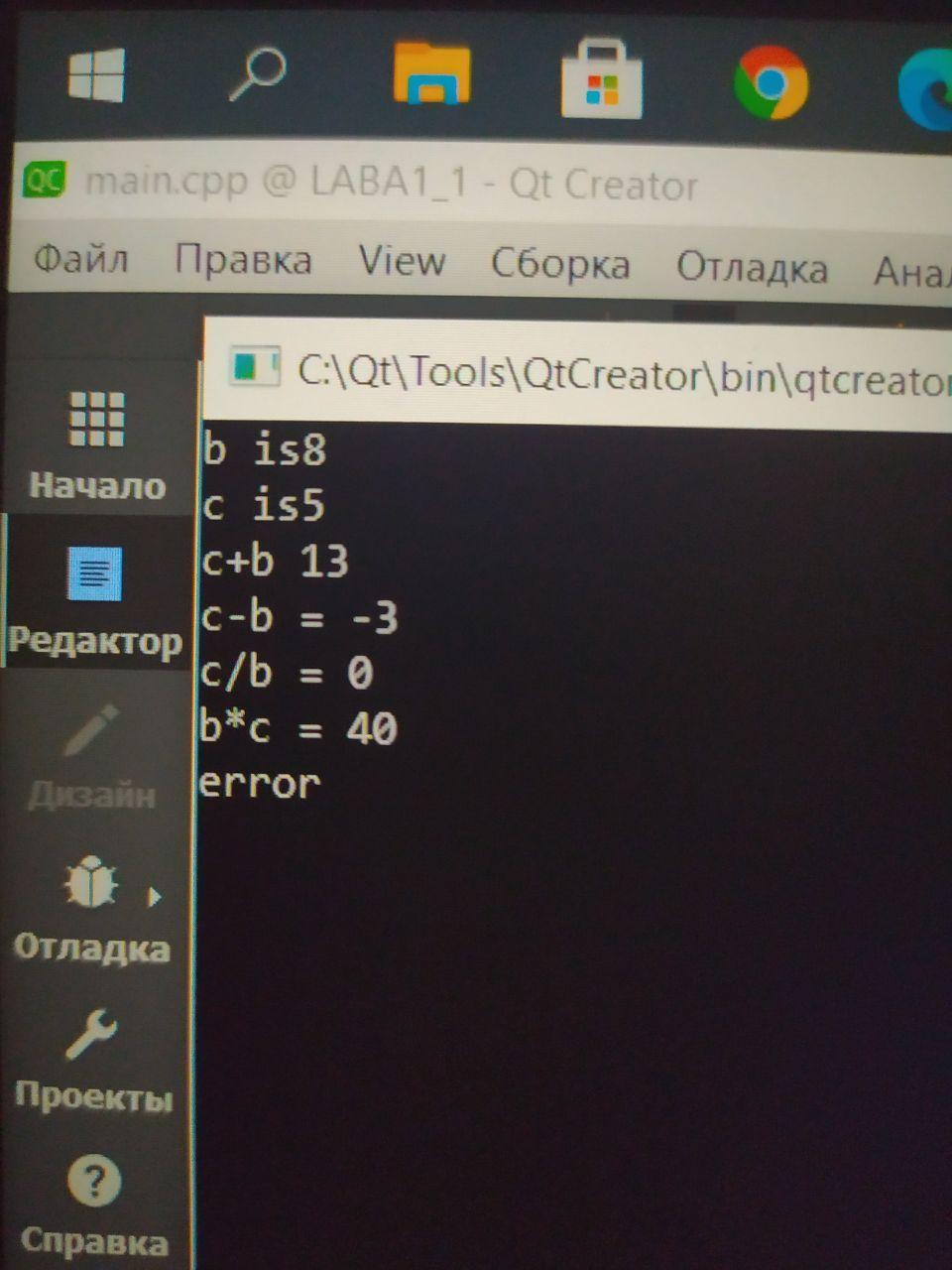
std::cout << "c/b = " << n3.getA() << std::endl;

n3=c\*b;

std::cout << "b\*c = " << n3.getA() << std::endl;

n3=n4\*b; }

**Результат виконання програми:**



**Завдання 2.** Для класу, який розробленого згідно індивідуального завдання лабораторної роботи № 2, визначити операції: - зчитування з потоку вводу sdt::cin; - виводу у потік std::cout. Перевірити роботу перевизначених функцій у функції main() за допомогою коду:

НазваКласуЗгідноВаріанта myObject; std::cin >> myObject; std::cout << myObject;

**Завдання 3.**\* Для заданого варіанта індивідуального завдання виконати перевантаження операцій для зручності роботи з об’єктами. При необхідності оголосіть певні операторні функції друзями класу

**Індивідуальне завдання:** Створити клас – двозв’язний список. Визначити необхідні конструктори, деструктор. Перевантажити потокові операції введення і виведення, операції + (додавання елемента в початок списку), += (додавання елемента в кінець списку), – (вилучення заданого елемента зі списку), ++ (перехід до наступного елемента), -- (перехід до попереднього елемента).

**Код програми(завдання 2 і 3):**

**Main.cpp:**

*#include <QCoreApplication>*

*#include<list.h>*

*List::****List****(const List & L)*

*{ m\_head = m\_tail = NULL;*

*m\_count = 0;*

*Elem \* temp = L.m\_head;*

*while(temp != 0)*

*{*

*addTail(temp->data);*

*temp = temp->next;*

*}*

*m\_position = L.m\_position;*

*}*

*List::****List****()*

*{*

*m\_head = m\_tail = 0;*

*m\_count = 0; m\_position = 0;*

*}*   
*bool List::operator+=(int n)*

*{ Elem \* temp = new Elem;*

*temp->next = 0;*

*temp->data = n;*

*temp->prev = m\_tail;*

*if(m\_tail != 0)*

*m\_tail->next = temp;*

*if(m\_count == 0)*

*m\_head = m\_tail = temp;*

*else*

*m\_tail = temp;*

*m\_count++;*

*return true;*

*}*

*bool List::operator+(int n)*

*{*   
 *Elem \* temp = new Elem;*

*temp->prev = 0;*

*temp->data = n;*

*temp->next = m\_head;*

*if(m\_head != 0)*

*m\_head->prev = temp;*

*if(m\_count == 0)*

*m\_head = m\_tail = temp;*

*Else*

*m\_head = temp;*

*m\_count++;*

*return true;*

*}*

*bool List::operator--()*

*{*

*if(m\_position > 0)*

*m\_position--;*

*return true;*

*}*

*bool List::operator++(){*

*if(m\_position < m\_count)*

*m\_position++;*

*return true;*

*}*

*void List::operator-(int pos)*

*{ del(pos); }*

*void List::****addTail****(int n)*

*{*

*Elem \* temp = new Elem;*

*temp->next = 0;*

*temp->data = n;*

*temp->prev = m\_tail;*

*if(m\_tail != 0) m\_tail->next = temp;*

*if(m\_count == 0)*

*m\_head = m\_tail = temp;*

*else*

*m\_tail = temp; m\_count++; }*   
*void List::****del****(int pos)*

*{*   
 *if(pos == 0)*

*{*

*std::cout << "Input position: ";*

*std::cin >> pos;*

*}*

*if(pos < 1 || pos > m\_count)*

*{*

*std::cout << "Incorrect position !!!\n";*

*return;*

*}*

*int i = 1;*

*Elem \* Del = m\_head;*

*while(i < pos)*

*{*

*Del = Del->next;*

*i++;*

*}*

*Elem \* PrevDel = Del->prev;*

*Elem \* AfterDel = Del->next;*

*if(PrevDel != 0 && m\_count != 1)*

*PrevDel->next = AfterDel;*

*if(AfterDel != 0 && m\_count != 1)*

*AfterDel->prev = PrevDel;*

*if(pos == 1)*

*m\_head = AfterDel;*

*if(pos == m\_count)*

*m\_tail = PrevDel;*

*delete Del;*

*m\_count--;*

*}*

*std::ostream& operator<<(std::ostream& out, const List &list)*

*{*

*if(list.m\_count != 0)*

*{*

*List::Elem \* temp = list.m\_head;*

*out << "( ";*

*while(temp->next != 0)*

*{*

*out << temp->data << ", ";*

*temp = temp->next;*

*}*   
 *out << temp->data << " )\n";*

*}*

*out << "position is " << list.m\_position; return out;*

*}*   
*std::istream& operator>>(std::istream& in, List &list)*

*{*   
 *int count, n;*

*list.m\_head = list.m\_tail = 0;*

*list.m\_count = 0; list.m\_position = 0;*

*std::cout << "Enter a number of things in your list?";*

*in >> count;*

*for(int i = 1;i <= count;i++ )*

*{ std::cout << "Enter element:" ;*

*std::cin >> n;*

*list.addTail(n);*

*}*

*return in;*

*}*

*List::~****List****()*

*{*

*while(m\_count != 0) del(1);*

*}*

*int* ***main****(int argc, char \*argv[])*

*{*

*QCoreApplication a(argc, argv);*

*List students;*   
 *std::cin >> students; students.addTail(8);*

*std::cout << students; students-1;*

*students+10;*

*++students;*

*students+=11;*

*std::cout << students; return a.exec();*

*}*

*“list.h”:*

#define LIST\_H

#include<iostream>

class **List**

{

private:

struct **Elem**

{

int data;

Elem \*next, \*prev;

};

Elem \*m\_head, \*m\_tail;

int m\_count, m\_position;

public:

**List**();

**List**(const List & L);

bool operator+=(int n); //новий елемент в хвіст

bool operator+(int n);//добавлення елемента в голову void operator-(int pos); // видалення елемента

bool operator--(); // позиція -

bool operator++();// позиція +

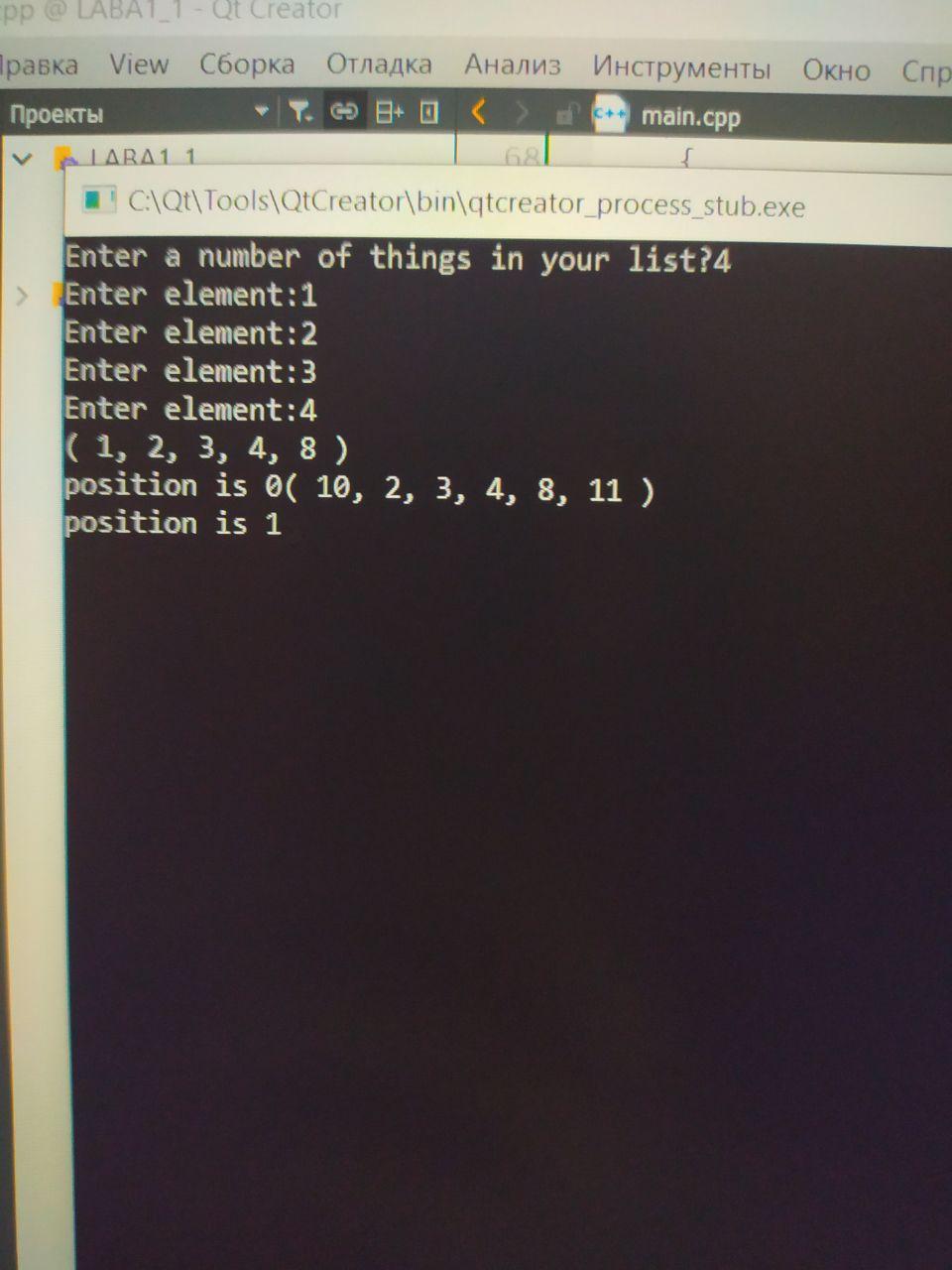
void **del**(int pos); // видалення елемента

void **addTail**(int n);//додати елемент в хвіст

friend std::ostream& operator<<(std::ostream& out, const List &list);

friend std::istream& operator>>(std::istream& in, List &list);   
 ~**List**(); };   
#endif // LIST\_H

**Результат виконання програми:**



**Висновок:** Я вивчила перевантаження операцій класу.