Міністерство освіти і науки, молоді та спорту України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 3**

«Створення та використання класів»

**з дисципліни «Об’єктно-орієнтоване програмування»**

**Лектор:**

доцент кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-14

Губик А. С.

**Прийняв:**

доцент кафедри ПЗ

Дяконюк Л. М.

«\_\_\_» \_\_\_\_\_\_ 2023 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема роботи:** Створення та використання класів.

**Мета роботи:** Навчитися створювати класи, використовувати конструктори для ініціалізації об’єктів, опанувати принципи створення функцій-членів. Навчитися використовувати різні типи доступу до полів та методів класів.

**Теоретичні відомост**і

Ідея класів має на меті дати інструментарій для відображення будови об’єктів реального світу - оскільки кожен предмет або процес має набір характеристик (відмінних рис) іншими словами, володіє певними властивостями і поведінкою. Програми часто призначені для моделювання предметів, процесів і явищ реального світу, тому в мові програмування зручно мати адекватний інструмент для представлення цих моделей.

Клас є типом даних, який визначається користувачем. У класі задаються властивості і поведінка будь-якого предмету або процесу у вигляді полів даних (аналогічно до того як це є в структурах) і функцій для роботи з ними. Створюваний тип даних володіє практично тими ж властивостями, що і стандартні типи.

Конкретні величини типу даних «клас» називаються екземплярами класу, або об'єктами.

Об’єднання даних і функцій їх обробки з одночасним приховуванням непотрібної для використання цих даних інформації називається інкапсуляцією (encapsulation). Інкапсуляція підвищує ступінь абстракції програми: дані класу і реалізація його функцій знаходяться нижче рівня абстракції, і для написання програми з використанням вже готових класів інформації про них (дані і реалізацію функцій) не потрібно. Крім того, інкапсуляція дозволяє змінити реалізацію класу без модифікації основної частини програми, якщо інтерфейс залишився тим самим (наприклад, при необхідності

змінити спосіб зберігання даних з масиву на стек). Простота модифікації, як уже неодноразово зазначалося, є дуже важливим критерієм якості програми.

**Завдання**

1. Створити клас відповідно до варіанту.

2. При створенні класу повинен бути дотриманий принцип інкапсуляції.

3. Створити конструктор за замовчуванням та хоча б два інших конструктори для початкової ініціалізації об’єкта.

4. Створити функції-члени для задавання та зчитування значень полів (getters/setters)

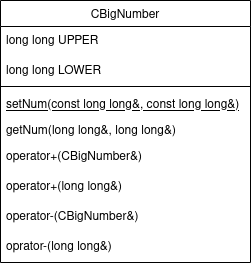
5. Створити інші функції члени згідно з варіантом.

6. Продемонструвати можливості класу написавши для нього модульні тести для Google Test.

7. У звіті до лабораторної намалювати UML-діаграму класу, яка відповідає варіанту.

8. Звіт має містити тексти програм, тестів та результат виконання тестів.

**Діаграма класу**



**Код**

**test.cpp:**

#include <gtest/gtest.h>

#include <climits>

#include "BigNumber.h"

TEST(int128, Adding){

CBigNumber b, a(1, LLONG\_MAX);

long long l, h;

b = a + CBigNumber(1, 10);

b.getNum(h, l);

EXPECT\_EQ(h, 3);

EXPECT\_EQ(l, 9);

b = a + 10;

b.getNum(h, l);

EXPECT\_EQ(h, 2);

EXPECT\_EQ(l, 9);

b = a + CBigNumber(LLONG\_MAX, 1);

b.getNum(h, l);

EXPECT\_EQ(h, -LLONG\_MAX);

EXPECT\_EQ(l, 0);

}

TEST(int128, dis){

CBigNumber b, a;

long long l, h;

a.setNum(1, -2);

b = a - LLONG\_MAX;

b.getNum(h, l);

EXPECT\_EQ(h, 0);

EXPECT\_EQ(l, 0);

b = a - CBigNumber(0, LLONG\_MAX);

b.getNum(h, l);

EXPECT\_EQ(h, 0);

EXPECT\_EQ(l, 0);

}

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

{

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**function.h:**

class CBigNumber{

long long UPPER;

long long LOWER;

public:

CBigNumber() : UPPER(0), LOWER(0){};

CBigNumber(const long long &rhs) : UPPER(0), LOWER(rhs){};

CBigNumber(const long long &upper\_rhs, const long long &lower\_rhs)

: UPPER(upper\_rhs), LOWER(lower\_rhs){};

CBigNumber operator+(const CBigNumber &rhs) const;

CBigNumber operator+(const long long &rhs) const;

CBigNumber operator-(const CBigNumber &rhs) const;

CBigNumber operator-(const long long &rhs) const;

void setNum(const long long &upper, const long long &lower);

void getNum(long long &upper, long long &lower);

};

**function.cpp:**

#include "BigNumber.h"

#include <climits>

CBigNumber CBigNumber::operator+(const CBigNumber &rhs) const{

long long tmp = LOWER + rhs.LOWER;

return CBigNumber(UPPER + rhs.UPPER + (tmp < LOWER),

-(LLONG\_MAX + 1) \* (tmp < LOWER) + LOWER + rhs.LOWER);

}

CBigNumber CBigNumber::operator+(const long long &rhs) const{

long long tmp = LOWER + rhs;

return CBigNumber(UPPER + (tmp < LOWER),

-(LLONG\_MAX + 1) \* (tmp < LOWER) + LOWER + rhs);

}

CBigNumber CBigNumber::operator-(const CBigNumber &rhs) const{

long long tmp = LOWER - rhs.LOWER;

return CBigNumber(UPPER - rhs.UPPER - (tmp > LOWER) ,

(LLONG\_MAX) \* (tmp > LOWER) + LOWER - rhs.LOWER + 2 \*(tmp > LOWER));

}

CBigNumber CBigNumber::operator-(const long long &rhs) const{

long long tmp = LOWER - rhs;

return CBigNumber(UPPER - (tmp > LOWER),

(LLONG\_MAX) \* (tmp > LOWER) + LOWER - rhs + 2 \*(tmp > LOWER));

}

void CBigNumber::setNum(const long long &upper, const long long &lower){

UPPER = upper;

LOWER = lower;

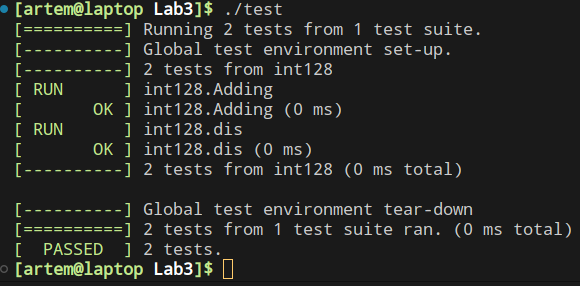
}

void CBigNumber::getNum(long long &upper, long long &lower){

upper = UPPER;

lower = LOWER;

}



**Висновок**

Класи шось там.