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 № 5**

«Робота з динамічною пам’яттю»

**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 – 2023

**Тема роботи:** Робота з динамічною пам’яттю

**Мета роботи:** Навчитися виділяти місце під об’єкти динамічно. Навчитися створювати та використовувати  
конструктор копіювання та переміщення, перевантажувати оператор присвоєння та переміщення.  
Ознайомитися з принципами створення та функціонування деструкторів.

**Теоретичні відомост**і

Перевизначення операторів

С++ підтримує спеціальні засоби, які дозволяють перевизначити вже існуючі оператори. Наприклад,

для оператора + можна ввести своє власне визначення, яке реалізує операцію додавання для об’єктів

певного класу. Фактично перевизначення для операцій існує і в мові С. Так, операція + може

використовувати як об’єкти типу int, так і об’єкти типу float. С++ розширює цю ідею.

Для визначення оператора використовується функція, що вводиться користувачем. Тип функції

визначається іменем класу, далі записується ключове слово operator, за яким слідує назва оператора,

в круглих дужках дається перелік параметрів, серед яких хоча б один типу клас.

Функція operator+ повертає результат типу complex та має параметр типу complex. Функції-операції

мають бути нестатичними функціями-членами класу або функціями поза межами класу та мати

мінімум один аргумент типу класу, посилання на клас, перелічення (enumeration) чи посилання на

перелічення.

За виключенням операції присвоєння всі перевизначені оператори наслідуються.

Дружні функції

Дружньою функцією класу називається функція, яка сама не є членом класу, але має повні права на

доступ до закритих та захищених елементів класу. Оскільки така функція не є членом класу, то вона

не може бути викликана з допомогою операторів (.) та (->). Дружня функція класу викликається

звичайним способом. Для опису дружньої функції використовується ключове слово friend.

Статичні змінні класу

До тепер вважалось, що дані в кожному об’єкті є власністю саме цього об’єкту та не використовуються

іншими об’єктами цього класу. Та іноді виникає потреба мати доступ до спільних даних для усіх

екземплярів даного класу (або й ззовні). Наприклад, необхідно з’ясувати скільки об’єктів даного класу

було створено на даний момент та скільки з них існує. Статичні змінні-члени досяжні для всіх

екземплярів класу. Це є компроміс між глобальними даними, які досяжні всім елементам програми, та

даними, що досяжні тільки об’єктам даного класу.

Статична змінні створюється в одному екземплярі для всіх об’єктів даного класу. Розглянемо приклад.

Об’єкт класу Cat включає статичну змінну-член HowManyCats (скільки котів). Ця змінна нараховує

кількість об’єктів класу Cat, що створені під час виконання програми.

**Завдання**

**Варіант 3**

Клас CDeposit має містити чисто віртуальну функцію (повертає суму виплат після  
заданої кількості місяців):

virtual double GetPayoutAfterNMonths(unsigned int uiMonthCount) const = 0;

Класи CFixedDeposit, CAccumulativeDeposit та CVIPDeposit мають реалізувати цю

функцію відповідно до своїх правил.  
Написати функцію, яка отримуватиме масив вказівників на базовий клас – CDeposit –  
та, викликаючи GetPayoutAfterNMonths () обчислюватиме загальну суму виплат через  
задану кількість місяців.

**Код**

**test.cpp:**

#include <gtest/gtest.h>

#include <climits>

#include "BigNumber.h"

TEST(int128, Adding){

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

long long h;

unsigned long long l;

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(ULLONG\_MAX, 1);

b.getNum(h, l);

EXPECT\_EQ(h, 1);

EXPECT\_EQ(l, 0);

}

TEST(int128, dis){

CBigNumber b, a;

long long h;

unsigned long long l;

a.setNum(1, 0);

b = a - ULLONG\_MAX - 2;

b.getNum(h, l);

EXPECT\_EQ(h, 0);

EXPECT\_EQ(l, ULLONG\_MAX);

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

b.getNum(h, l);

EXPECT\_EQ(h, 1);

EXPECT\_EQ(l, 1);

}

TEST(int128, compare){

CBigNumber c(0, ULLONG\_MAX), b(1, 0),

a(1, ULLONG\_MAX), a1(1, ULLONG\_MAX);

EXPECT\_TRUE(a > b);

EXPECT\_TRUE(a > c);

EXPECT\_FALSE(a < b);

EXPECT\_FALSE(a < c);

EXPECT\_TRUE(a == a1);

EXPECT\_FALSE(a == b);

EXPECT\_FALSE(a == c);

}

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

{

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

return RUN\_ALL\_TESTS();

}

**function.h:**

#include <iostream>

class CBigNumber{

long long upper;

unsigned long long lower;

public:

CBigNumber();

explicit CBigNumber(const unsigned long long &rhs);

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

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

void setNum(const unsigned long long &lower);

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

CBigNumber operator+(const CBigNumber &rhs);

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

CBigNumber operator-(const CBigNumber &rhs);

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

friend bool operator> (const CBigNumber &lhs, const CBigNumber &rhs);

friend bool operator< (const CBigNumber &lhs, const CBigNumber &rhs);

friend bool operator==(const CBigNumber &lhs, const CBigNumber &rhs);

friend std::istream &operator>>(std::istream &out, CBigNumber &rhs);

friend std::ostream &operator<<(std::ostream &out, const CBigNumber &rhs);

};

**function.cpp:**

#include "BigNumber.h"

#include <climits>

CBigNumber::CBigNumber() : upper(0), lower(0){};

CBigNumber::CBigNumber(const unsigned long long &rhs) : upper(0), lower(rhs){};

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

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

void CBigNumber::setNum(const long long &u, const unsigned long long &l){

upper = u;

lower = l;

}

void CBigNumber::setNum(const unsigned long long &l){

upper = 0;

lower = l;

}

void CBigNumber::getNum(long long &u, unsigned long long &l) const{

u = upper;

l = lower;

}

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

unsigned long long tmp = lower + rhs.lower;

return CBigNumber(upper + rhs.upper + (tmp < rhs.lower), tmp);

}

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

unsigned long long tmp = lower + rhs;

return CBigNumber(upper + (tmp < rhs), tmp);

}

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

unsigned long long tmp = lower - rhs.lower;

return CBigNumber(upper - rhs.upper - (tmp > rhs.lower) , tmp);

}

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

unsigned long long tmp = lower - rhs;

return CBigNumber(upper - (tmp > rhs), tmp);

}

bool operator>(const CBigNumber &lhs, const CBigNumber &rhs){

if(lhs.upper == rhs.upper)

return (lhs.lower > rhs.lower);

return (lhs.upper > rhs.upper);

}

bool operator<(const CBigNumber &lhs, const CBigNumber &rhs){

if(lhs.upper == rhs.upper)

return (lhs.lower < rhs.lower);

return (lhs.upper < rhs.upper);

}

bool operator==(const CBigNumber &lhs, const CBigNumber &rhs){

return(lhs.upper == rhs.upper && lhs.lower == rhs.lower);

}

std::istream &operator>>(std::istream &in, CBigNumber &rhs){

in >> rhs.upper >> rhs.lower;

return in;

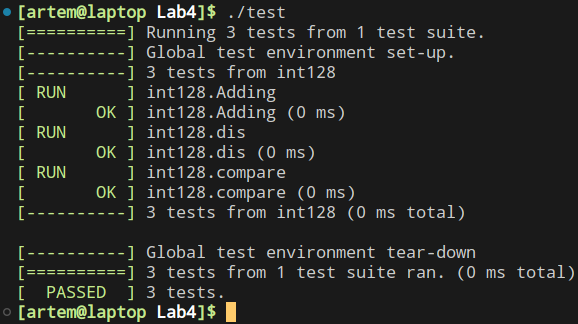
}

std::ostream &operator<<(std::ostream &out, const CBigNumber &rhs){

out << rhs.upper << rhs.lower;

return out;

}



**Висновок**

Динамічна пам’ять допомагає.