Міністерство освіти і науки

Національний університет “Львівська політехніка”



**ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ**

**Інструкція до лабораторної роботи № 6**

з курсу “Алгоритмізація та програмування. Частина 1”

для студентів

спеціальності 122 “Комп’ютерні науки та інформаційні технології”

спеціалізації “Системна інженерія (інтернет речей)”

Затверджено

на засіданні кафедри

“Комп’ютеризовані

системи автоматики”

Протокол № 17 від 08.06.2017 р.

**Львів 2017**

Перевантаження операцій. Інструкція до лабораторної роботи №6 з курсу “Алгоритмізація та програмування ч.1” для студентів спеціальності 122 “Комп’ютерні науки та інформаційні технології” спеціалізації “Системна інженерія (інтернет речей)” / Укл.: У.Ю. Дзелендзяк, З.Є. Верес – Львів: НУЛП, 2017.- 26 с.

Укладачі: У.Ю. Дзелендзяк, к.т.н., доцент

З.Є. Верес, к.т.н., асистент

Відповідальний за випуск:

А.Й. Наконечний, д.т.н., професор

Рецензент: І.М. Бучма, д.т.н., професор

**Мета роботи:** вивчити основні принципи перевантаження операцій в алгоритмічній мові С++, навчитися перевантажувати унарні та бінарні операції для конкретного класового типу.

1. **Короткі теоретичні дані**
   1. **Операторні функції**

Спочатку необхідно відзначити, що не всі символи операцій можна перевантажити. Нижче перераховані оператори, дозволені до перевантаження:

+ - \* / % ^ &

| ~ ! = < > +=

-= \*= /= %= ^= &= |=

<< >> >>= <<= == != <=

>= && || ++ -- ->\* ,

-> [] () new new[] delete delete[]

Існують також оператори, заборонені до перевантаження. Зміна їх змісту зруйнувала б логіку програми. До таких операторів належать :: (оператор дозволу області видимості), . (“крапка” — оператор доступу до члена класу), ? : (тернарний оператор), .\* (доступ до розіменованого вказівника-члена класу), sizeof, typeid, static\_cast, dynamic\_cast, const\_cast і reinterpret\_cast. Крім того, не рекомендується перевантажувати логічні оператори && та ||, оскільки на їхні перевантажені версії не поширюється правило скорочених обчислень логічних виразів. (Нагадаємо, що це правило полягає в наступному: якщо на деякому етапі значення усього виразу стає визначеним, подальші обчислення припиняються.)

Синтаксис операторних функцій має вигляд:

тип\_значення\_що\_повертається operator символ\_операції(параметри)

{

}

Наприклад, операторна функція, що перевантажує операцію +, називається operator+().

Операторні функції повинні мати прямий доступ до членів класу. Отже, необхідно, щоб вони були або членами класу, або дружніми функціями.

* 1. **Перевантаження унарних операторів за допомогою  
      функцій-членів**

Оператори можуть бути унарними і бінарними. Унарний оператор має один операнд, а бінарний — два. Нагадаємо, що до унарних операторів, які перевантажуються, належать такі оператори, як +, -, ++, - -, &, ~, |, !. До бінарних операторів, що перевантажуються, належать всі інші оператори, перераховані вище.

Операторні функції-члени, що перевантажують унарний оператор, мають одну особливість: їх операнди передаються неявно за допомогою вказівника this. Отже, така функція-член класу не має явних параметрів.

* + 1. **Унарні оператори «плюс» та «мінус»**

Розглянемо приклад перевантаження унарного мінуса.

#include <stdio.h>

class TComplex {

double Re; double Im; public:

TComplex(double x, double y):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){} void print();

TComplex operator-() {Re = -Re; Im = -Im; return \*this;}

};

int main()

{

TComplex z(1,1),u(0,0); z.print () ; u=- z;

u.print();

return 0;

}

void TComplex::print()

{

printf("%lf + i\*%lf\n", Re, Im);

}

Відзначимо, що операторна функція може повертати об'єкт класу (точніше, розіменований вказівник this) чи посилання на об'єкт, а може і нічого не повертати. Вибираючи тип значення, що повертається, необхідно керуватися здоровим глуздом. Якщо унарний оператор використовується усередині виразів, то операторна функція повинна повертати об'єкт чи посилання. Якщо ж оператор використовується ізольовано, функція може нічого не повертати.

Рядок програми u = - z;

можна переписати в еквівалентному виді. u = z.operator-();

Цей рядок демонструє, що виклик операторної функції operator- () здійснюється об'єктом z.

Варто мати на увазі, що хоча зміст оператора перевантажувати можна, його природу змінювати заборонено. Це значить, що унарний оператор не можна перевантажити як бінарний і навпаки.

Розглянемо найбільш важливі приклади унарних операторів.

* + 1. **Оператори інкремента і декремента**

У мові С++ передбачено дві форми операторів інкремента і декремента: префіксна і постфіксна. Для того щоб розрізняти їх, використовується звичайний механізм перевантаження функцій — вводиться фіктивний цілочисловий параметр. Наприклад, для класу TComplex прототипи відповідних операторних функцій можуть виглядати в так:

TComplex operator ++(); // Префіксна форма

ТСоmрlех ореrаtоr++(іnt); // Постфіксна форма

ТСоmрlех ореrаtоr--(); // Префіксна форма

ТСоmрlех ореrаtоr--(іnt); // Постфіксна форма

Покажемо це на конкретному прикладі:

#include <stdio.h>

class TComplex {

double Re; double Im;

public:

TComplex(double x, double y):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){} void print();

TComplex& operator++()

{

++ Re;

++ Im;

printf("Префіксна форма ++ \n");

return \*this;

}

const TComplex operator++(int i)

{

++ Re;

++ Im;

printf("Постфіксна форма ++ %d\n",i);

return \*this;

}

TComplex& operator--()

{

--Re;

--Im;

printf("Префіксна форма -- \n");

return \*this;

}

const TComplex operator--(int i)

{

--Re;

--Im;

printf("Постфіксна форма -- %d\n",i);

return \*this;

}

};

int main()

{

TComplex z(1,1);

++ z;

z.print();

z++;

z.print () ;

--z;

z.print () ;

z--;

z.print () ;

return 0;

}

void TComplex::print ()

{

printf("%lf + i\*%lf\n", Re, Im);

}

Якщо символ операції ++ стоїть перед операндом, викликається операторна функція operator++ (), якщо після — операторна функція operator++(int i). Змінна і відіграє роль прапора, що повідомляє компілятору, що дана функція перевантажує постфіксну форму оператора інкремента і декремента.

Незважаючи на те що програміст може вільно трактувати перевантажений оператор, прагнучи зберегти аналогію з його вбудованими аналогами, необхідно дотримуватись визначених правил. Наприклад, як відомо, оператор інкрементації цілих чисел повертає посилання на неконстантний об'єкт. Ця властивість повинна зберігатися і при перевантаженні.

TComplex& operator++()

{

++Re;++ Im;

ргіntf("Префіксна форма ++ \n") ;

return \*this;

}

У той же час постфіксний оператор повертає константне значення. Саме тому в С++ неможливі вирази x+++ + . Цілком природно зажадати, щоб перевантажений оператор мав таку ж властивість.

const TComplex operator++(int і)

{

++Re; ++Im;

printf("Постфіксна форма ++ %d\n",i);

return \*this;

}

Крім того, оскільки у вихідному варіанті постфіксний оператор інкрементації реалізується через префіксний, це також бажано врахувати при перевантаженні. У цьому випадку реалізація операторної функції ++ для постфіксної форми може виглядати так:

const TComplex operator++(int і)

{

TComplex Z;

++\*this;

printf("Постфиксна форма ++ %d\n",i);

return Z;

}

* + 1. **Унарні оператори !, &, |, ~**

Оператори заперечення (!), взяття адреси (&) і побітового заперечення (~) допускають перевантаження, але не мають універсальних альтернатив, що варто було б реалізувати. Їх можна перевантажувати, наприклад, для підвищення наочності програми. Скажемо, за допомогою оператора ! можна позначати операцію звертання матриці, а за допомогою символу ~ — її транспонування. Щоправда, застосування тильди закріплене за деструкторами, тому варто виявляти обережність, щоб не створити плутанину. У будь-якому випадку зміст перевантаження операторів залежить від конкретної задачі.

* + 1. **Перевантаження оператора** ->

Оператор посилання на член об'єкта є унарним. Операторна функція, що перевантажує його, має вигляд:

об'єкт -> елемент

Цей запис еквівалентний наступному виразу:

об'єкт.operator->(елемент) ;

Функція operator-> () повинна бути нестатичним членом класу. Як параметр вона одержує об'єкт класу чи посилання на нього, повертаючи вказівник this на об'єкт, де виконується виклик, або посилання на об'єкт будь-якого іншого класу, у якому визначений оператор ->. ЇЇ зручно використовувати в контейнерних класах, що містять усередині себе вказівник на інший клас. Основний зміст перевантаження оператора -> полягає в додатковій функціональності, що розширює можливості звичайних вказівників.

Наприклад, у наведеній нижче програмі функція operator->() веде підрахунок посилань на кожен об'єкт класу.

#include <stdio.h>

class TClass {

int n;

int counter; public:

TClass(int x):n(x),counter(0) { }

TClass\* operator->();

int get(void) { return n;}

int ref(void) { return counter; }

};

TClass\* TClass::operator ->()

{

counter++;

return this;

}

int main()

{

TClass a(1), b(2);

printf("n = %d \n",a->get());

printf("n = %d \n",b->get());

printf("n = %d \n",a->get());

printf("counter = %d \n",a->ref());

printf("counter = %d \n",b->ref());

return 0;

}

Зверніть увагу на те, що функція operator- >() у даному прикладі повертає значення типу TClass\*. Розглянемо тепер програму, у якій створюється контейнер, що містить масив вказівників на об'єкти класу TClass.

#include <stdio.h>

class TClass

{

int n;

int counter;

public:

TClass(int x):n(x),counter(0) { }

TClass\* operator->();

int get(void) { return n;}

int ref(void) { return counter; }

};

TClass\* TClass::operator ->()

{

counter++;

return this;

}

class TContainer

{

public:

int m;

TClass\* p[];

TContainer(int k, TClass\* q[])

{

m = k;

for(int i=0; i<m; i++)p[i]=q[i];

}

TContainer\* operator->();

};

TContainer\* TContainer::operator ->()

{

return this;

};

int main()

{

int m = 2;

TClass a(1), b(2);

TClass\* p[]={&a, &b};

TContainer c(m,p);

for(int i=0; i<m; i++) printf("%ld \n",c->p[i]->get());

return 0;

}

Оскільки й у класі TContainer, і в класі TClass міститься перевантажена версія оператора ->, ми можемо скористатися ланцюжком операторів   
c->p[i]->get (), що забезпечує перегляд умісту контейнера.

* 1. **Перевантаження бінарних операторів за допомогою   
     функцій-членів**

Бінарний оператор має два операнди. Його виклик виконується об'єктом, розташованим у лівій частині оператора. Отже, бінарний оператор a+b еквівалентний такому оператору. a.operator+(b) Таким чином, бінарна операторна функція-член повинна мати тільки один параметр, що задає другий операнд Вказівник this на перший операнд вона одержує неявно. Для того, щоб бінарну операторну функцію можна було застосовувати всередині виразів, необхідно, щоб вона повертала об'єкт свого класу. Розглянемо приклад.

Перевантаження бінарного оператора +

#include <stdio.h>

class TComplex

{

double Re;

double Im;

public:

TComplex(double x, double y):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){}

void print();

TComplex operator+(TComplex z)

{

TComplex w(0,0);

w.Re = Re+z.Re;

w.Im = Im+z.Im;

return w;

}

};

int main()

{

TComplex u(1,1),v(2,2),z(0,0);

z=u+v;

z.print();

return 0;

}

void TComplex::print()

{

printf("%lf + i\*%lf\n", Re, Im);

}

У цій програмі сума комплексних чисел u і v привласнюється об'єкту z. Виклик операторної функції operator+() здійснюється з об'єкта u.

Оскільки сума двох об'єктів класу TComplex є об'єктом цього ж класу, не обов'язково створювати новий об'єкт і записувати в нього результат підсумовування. Можна скористатися наступною синтаксичною конструкцією. (u+v).print(); У цьому випадку функція print() буде викликатися тимчасовим об'єктом, створеним операторної функцією operator+(). Цікаво, що цей механізм допускає вживання безглуздих виразів. Наприклад, компілятор не заперечує проти такого оператора. (u+v)=z; Для того щоб запобігти цьому, оператор + повинний повертати константний об'єкт. Крім того, бажано, щоб він не змінював другий операнд. Отже, параметр повинен бути константним. І останнє: тому що об'єкт може бути досить громіздким, його варто передавати за значенням. Отже, одержуємо наступну реалізацію.

const TComplex operator+(const TComplex& z)

{

TComplex w(0,0);

w.Re = Re+z.Re;

w.Im = Im+z.Im;

return w;

}

* 1. **Перевантаження унарних операторів за допомогою   
     дружніх функцій**

У деяких ситуаціях перевантажені оператори повинні бути членами класу. До них належать оператори =, [], ->, new і delete. В інших випадках програміст повинний керуватися здоровим глуздом. Перевантаження операторів у вигляді дружніх функцій нічим не відрізняється від перевантаження у вигляді членів класу, за одним виключенням — функція тепер не одержує неявний вказівник \*this. Отже, всі операнди повинні бути зазначені явно. Розглянемо приклад, у якому унарний оператор “мінус” перевантажується дружньою функцією.

#include <stdio.h>

class TComplex

{

double Re;

double Im;

public:

TComplex(double x, double y):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){}

friend void print(TComplex z);

friend TComplex operator-(TComplex z);

};

int main()

{

TComplex z(1,1),u(0,0);

print(z);

u=-z;

print(u);

return 0;

}

void print(TComplex z)

{

printf("%lf + i\*%lf\n", z.Re, z.Im);

}

TComplex operator-(TComplex z)

{

TComplex w(0,0);

w.Re = -z.Re;

w.Im = -z.Im;

return w;

}

Як бачимо, на відміну від попередніх варіантів, унарний мінус перевантажується операторною функцією, що має один операнд. Отже, у тілі функції необхідно створити локальний об'єкт, а потім змінити його зміст відповідно до визначення оператора. До недоліку такого підходу варто віднести додаткові витрати пам'яті і часу при створенні локальних об'єктів.

**1.5. Перевантаження бінарних операторів за допомогою  
 дружніх функцій**

Якщо стандарт не накладає особливих обмежень, програміст може сам вибирати спосіб перевантаження. Однак у деяких ситуаціях функції-члени можуть виявитися незручними. Наприклад, у нашому класі TComplex визначений оператор +, у якого і лівий, і правий операнди є об'єктами класу TComplex. У деяких випадках перевантаження арифметичних операторів за допомогою функцій-членів може привести до небажаних ефектів. Розглянемо наступну реалізацію.

Перевантаження оператора “плюс” за допомогою функції-члена:

#include <stdio.h>

class TComplex

{

double Re;

double Im;

public:

TComplex(double x, double y=0):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){}

void print();

const TComplex operator+(const TComplex& z)

{

TComplex w(0,0);

w.Re = Re+z.Re;

w.Im = Im+z.Im; return w;

}

};

int main()

{

TComplex u(1,1),v(2,2),z(0,0);

z = u + 1.0;

z.print();

return 0;

}

void TComplex::print()

{

printf("%lf + i\*%lf\n", Re, Im);

}

Оскільки другий параметр конструктора задається за замовчуванням, відкривається можливість неявного перетворення змінної типу double в об'єкт класу TComplex. Отже, вираз z = u + 1.0; є цілком коректним. Одержуємо результат: 2.000000 + i\*1.000000. Однак, особливості перевантаження операторів за допомогою функцій-членів порушують математичні правила: вираз 1.0+u невірний, оскільки лівим і правим операндами операторної функції operator+() повинні бути об'єкти класу TComplex. Якщо як правий об'єкт зазначена змінна типу double, вона буде неявно перетворена в об'єкт класу TComplex. Однак на об'єкт, що викликає функцію operator+(), це не поширюється. Компілятор просто повідомить, що для цієї ситуації в класі немає придатної функції-члена. Вирішити цю ситуацію просто — оператор потрібно перевантажити за допомогою дружньої функції.

Перевантаження оператора + за допомогою дружньої функції:

#include <stdio.h>

class TComplex

{

double Re;

double Im;

public:

TComplex(double x, double y):Re(x),Im(y){ }

TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}

~TComplex(){}

friend void print(TComplex z);

friend TComplex operator+(TComplex x, TComplex y);

};

int main()

{

TComplex u(1,1),v(2,2),z(0,0);

z=u+v;

print(z);

return 0;

}

TComplex operator+(TComplex x, TComplex y)

{

TComplex w(0,0);

w.Re = x.Re+y.Re;

w.Im = x.Im+y.Im;

return w;

}

void print(TComplex z)

{

printf("%lf + i\*%lf \n",z.Re,z.Im);

}

Тепер обидва параметри дружньої функції можуть неявно перетворюватися з типу double у тип TComplex, і комутативність додавання відновлена.

1. **Завдання**

Написати програму алгоритмічною мовою С++ згідно із завданням, отриманим від викладача за табл. 2.1. Реалізувати заданий класовий тип із заданими методами, використовуючи механізм перевантаження функцій та операцій, а також значення параметрів за замовчуванням. Написати фрагмент програми з використанням класового типу та заданих методів.

Таблиця 2.1

|  |  |  |
| --- | --- | --- |
| **№ п/п** | Класовий тип | Методи класу |
| 1 | Комплексне число | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без ініціалізації. Перевантажити операції +, -, \*, виводу <<, вводу >>, ==, >, < для об'єктів класу, а також дружні операції +, -= для масивів комплексних чисел. |
| 2 | Одномірний масив **A[N]** | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без ініціалізації. Перевантажити операції +, \*<тип>, виводу <<, вводу >>, =, == для об'єктів класу, а також дружні операції <тип>\*, -=<тип>. |
| 3 | Двомірний масив **A[N][N]** | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без ініціалізації. Перевантажити операції +, \*, =, виводу <<, вводу >> для об'єктів класу, а також дружні операції <тип>+, /=<тип>. |
| 4 | Динамічний одномірний масив **\*p** | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без неї. Визначити конструктор копій. Перевантажити операції +, =, виводу <<, вводу >> для об'єктів класу, а також дружні операції \*=<тип>, <тип>+. |
| 5 | Динамічний двомірний масив **\*\*p** | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без неї. Визначити конструктор копій. Перевантажити операції -, =, виводу <<, вводу >> для об'єктів класу, а також дружні операції <тип>\*, +=. |
| 6 | Прямокутник зі сторонами **a** і **b** | Конструктор повинен дозволяти ініціалізацію прямокутника і спрощену ініціалізацію квадрата. Визначення площі. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію \*=<тип>. |
| 7 | Трикутник із основою **a** , висотою **h** та кутами при основі α1і α2 | Конструктор повинен дозволяти спрощену ініціалізацію прямокутного та рівностороннього трикутників. Визначення площі, визначення периметра. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію /=<тип>. |
| 8 | Еліпс з радіусами **rx** і **ry** | Конструктор повинен дозволяти спрощену ініціалізацію кола. Визначення площі, визначення периметра. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію \*=<тип>. |
| 9 | Відрізок на координатній площині [**p1**;**p2**] | Конструктор повинен визначати довільний або одиничний відрізок в центрі координат. Визначення довжини відрізка. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію <<= для повороту відрізка на кут α. |
| 10 | N-кутник на координатній площині [**p1;p2;...;pN**] | Конструктор повинен визначати довільний N-кутник з можливістю ініціалізації. Перевантажити операції ==, виводу <<, вводу >>, а також дружню операцію >>= для переміщення N-кутника на [dx;dy]. |
| 11 | Відрізок в просторі [**p1;p2**]; | Конструктор повинен визначати довільний або одиничний відрізок в центрі координат. Визначення довжини відрізка. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію \*= для маштабування відрізка на [sx; sy]. |
| 12 | Трикутник в просторі [**p1;p2;p3**] | Конструктор повинен визначати довільний або рівносторонній трикутник в центрі координат. визначення периметра. Перевантажити операції ==, виводу <<, вводу >>, а також дружню операцію <<= для повороту трикутника на кут α. |
| 13 | Прямокутник в просторі [**p1;p2;p3;p4**]; | Конструктор повинен визначати довільний прямокутник або квадрат в центрі координат. визначення площі. Перевантажити операції ==, виводу <<, вводу >>, а також дружню операцію \*= для масштабування прямокутника на [sx; sy]. |
| 14 | N-кутник в просторі [**p1;p2;...;pN**] | Конструктор повинен визначати довільний N-кутник з можливістю ініціалізації. Перевантажити операції ==, виводу <<, вводу >>, а також дружню операцію >>= для переміщення N-кутника на [dx; dy]. |
| 15 | Символьний рядок | Конструктор повинен дозволяти створювати символьний рядок довільного розміру та при необхідності ініціалізувати його. Визначити конструктор копій. Перевантажити операції присвоєння =, порівняння ==, вилучення з рідка заданого символу -, виводу << і вводу >>. |
| 16 | Прямокутна матриця **n**x**m** | Конструктор повинен створювати матрицю заданої розмірності з можливістю ініціалізації. Визначити конструктор копій. Перевантажити операції +, =, ==, \*, виводу << і вводу >> для об'єктів класу. |
| 17 | Дата визначена в форматі дд.мм.рр. | Конструктор повинен дозволяти створювати об’єкти з ініціалізацією. Реалізувати операції =, ==, <, >, -, +=<int>, виводу << і вводу >> для об'єктів класу. |
| 18 | Поліном виду | Конструктор повинен дозволяти створювати поліном з ініціалізацією. Реалізувати операції =, ==, <, >, -, \*=<int>, виводу << і вводу >> для об'єктів класу. |
| 19 | Множина цілих чисел | Конструктор повинен дозволяти створювати множину з ініціалізацією. Реалізувати операції ==, &&(перетин множин), ||(об'єднання множин), +=(внесення нового елемента), -=(вилучення заданого елемента), виводу << і вводу >> для об'єктів класу. |
| 20 | Раціональний дріб виду | Конструктор повинен дозволяти створювати множину з ініціалізацією. Реалізувати операції ==, +, -, \*, /, виводу << і вводу >> для об'єктів класу. |
| 21 | Комплексне число, задане своїм модулем **A** і кутом α | Конструктор повинен дозволяти створювати об'єкти з ініціалізацією і без ініціалізації. Перевантажити операції +, -, \*, ==, >, <, виводу << і вводу >> для об'єктів класу. |
| 22 | Час визначений в форматі год.хв.сек. | Конструктор повинен дозволяти створювати об’єкти з ініціалізацією. Реалізувати операції =, ==, <, >, -, +=<int>, виводу << і вводу >> для об'єктів класу. |
| 23 | Прямокутний трикутник із гіпотенузою с, катетами a і b, кутами α1і α2 | Конструктор повинен дозволяти спрощену ініціалізацію прямокутного трикутника. Визначення площі, визначення периметра, знаходження тригонометричних функцій sin та cos Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію /=<тип>. |
| 24 | Куб із стороною а | Конструктор повинен дозволяти ініціалізацію куба і спрощену ініціалізацію квадрата. Визначення об’єму куба та площі поверхні куба. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію \*=<тип>. |
| 25 | Куля з центром в точці з координатами (a,b,c) та радіусом R | Конструктор повинен дозволяти ініціалізацію кулі. Визначення об’єму кулі та площі поверхні кулі. Вивід рівняння кулі. Перевантажити операції ==, <, >, виводу <<, вводу >>, а також дружню операцію \*/<тип>. |
| 26 | Множина дійсних чисел | Конструктор повинен дозволяти створювати множину з ініціалізацією. Реалізувати операції ==, &&(перетин множин), ||(об'єднання множин), +=(внесення нового елемента), -=(вилучення заданого елемента), виводу << і вводу >> для об'єктів класу. |
| 27 | Добуток послідовності виду | Конструктор повинен дозволяти створювати послідовність з ініціалізацією. Реалізувати операції =, ==, <, >, -, \*=<int>, виводу << і вводу >> для об'єктів класу. |
| 28 | Множина символьних значень | Конструктор повинен дозволяти створювати множину з ініціалізацією. Реалізувати операції ==, &&(перетин множин), ||(об'єднання множин), +=(внесення нового елемента), -=(вилучення заданого елемента), виводу << і вводу >> для об'єктів класу. |
| 29 | Двійковий 12-розрядний регістр | Конструктор повинен дозволяти ініціалізацію регістра довільним значенням або нулем. Перевантажити операції ++, ==, виводу <<, вводу >>, а також дружні операції -=<int>, <<=<int>. |
| 30 | Шіснадцятковий 8-розрядний регістр | Конструктор повинен дозволяти ініціалізацію регістра довільним значенням або нулем. Перевантажити операції ++, ==, виводу <<, вводу >>, а також дружні операції -=<int>, <<=<int>. |

1. **Робота в лабораторії**

1). Ввести в комп'ютер програму, написану мовою Cі++ згідно з отриманим завданням.

2). Відлагодити програму. При необхідності скоригувати програму у відповідності з виявленими логічними та синтаксичними помилками.

3). Остаточні версії програми та отримані результати занести у звіт з лабораторної роботи.

4). Здати звіт з лабораторної роботи.

### Зміст звіту

1. Номер і назва лабораторної роботи.
2. Повний текст завдання.
3. Список ідентифікаторів констант, змінних, функцій, використаних у програмі, та їх пояснення.
4. Текст програми.
5. Результати виконання програми.

#### Питання для самоконтролю

1. Які операції не можна перевантажити?

2. Як виглядає синтаксис операторних функцій?

3. Які вимоги висуваються до операторних функцій?

4. Які обмеження накладаються на застосування перевантажених операторів?

5. Назвіть особливість операторних функцій-членів, що перевантажують унарний оператор.

6. Назвіть дві форми перевантаження операторів інкремента і декремента.

7. Назвіть особливості перевантаження операторів інкремента і декремента. 8. Як перевантажується оператор посилання на член об'єкта?

9. Які обмеження накладаються на функцію operator->()?

10. Назвіть особливості бінарних операторних функцій-членів.

11. Які операторні функції обов’язково повинні бути членами класу?

12. Коли виникає необхідність перевантажених операцій як дружніх?

1. **Список лiтератури**
2. Стивен Прата. Язык программирования С++. Лекции и упражнения, 6-е издание. ИД «Вильямс», 2016. - 1248 с.
3. Р. Лафоре. Объектно-ориентированное программирование в С++.   
   4-е издание. СПб.: Питер, 2015. – 928 с.
4. Скотт Мейерс. Наиболее эффективное использование С++. 35 новых рекомендаций по улучшению ваших программ и проектов. Пер. с англ.– М.: ДМК Пресс., СПб.: Питер, 2016. – 298 с.
5. Страуструп Б. Дизайн и эволюция языка С++. Пер. с англ. – М.: ДМК Пресс., СПб.: Питер, 2016. – 446 с.

**Зміст**

1. Короткі теоретичні відомості………………………………… …………….3
   1. Операторні функції…………………………………………………………...3
   2. Перевантаження унарних операторів за допомогою функцій-членів.........4
      1. Унарні оператори «плюс» та «мінус»………………………………..4
      2. Оператори інкремента і декремента………………………………….5
      3. Унарні оператори !, &, |, ~…………………………………………….9
      4. Перевантаження оператора ->………………………………..………9
   3. Перевантаження бінарних операторів за допомогою функцій-членів…..12

1.4 Перевантаження унарних операторів за допомогою дружніх функцій…14

1.5. Перевантаження бінарних операторів за допомогою дружніх функцій..15

1. Завдання………………………………………….…………………………..18
2. Робота в лабораторії…………………………………………………………23
3. Зміст звіту………………………………………………………………….....23
4. Питання для самоконтролю…………………………………………………24
5. Список літератури…………………………………………………………...24

Навчальне видання

ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

ІНСТРУКЦІЯ ДО ЛАБОРАТОРНОЇ РОБОТИ № 6

З КУРСУ “АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ. ЧАСТИНА 1”

для студентів

спеціальності 122 “Комп’ютерні науки та інформаційні технології”

спеціалізації “Системна інженерія (інтернет речей)”

Укладачі: У.Ю. Дзелендзяк, к.т.н., доцент

З.Є. Верес, к.т.н., асистент

Львів: Національний університет “Львівська політехніка”