Лабораторна робота № 2

# ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

**Мета.** Отримати практичні навички створення абстрактних типів даних і перевантаження операцій в мові С ++.

**Основний зміст роботи.** Визначити і реалізувати клас – абстрактний тип даних. Визначити і реалізувати операції над даними цього класу. Написати і виконати програми повного тестування цього класу.

## Короткі теоретичні відомості.

*Абстрактний тип даних (АТД).* АТД – тип даних, який визначається тільки через операції, які можуть виконуватися над відповідними об'єктами безвідносно до способу подання цих об'єктів. АТД включає в себе абстракцію як через параметризацію, так і через специфікацію. Абстракція через параметризацію може бути здійснена так само, як і для процедур (методів); використанням параметрів там, де це має сенс. Абстракція через специфікацію досягається за рахунок того, що операції представляються як частина типу.

*Дружні функції.* Дружня функція – це функція, яка має доступ до закритих членів класу, наче вона сама є членом цього класу. У всіх інших аспектах дружня функція є звичайною функцією. Нею може бути, як звичайна функція, так і метод іншого класу. Для оголошення дружньої функції використовується ключове слово friend перед прототипом функції, яку ви хочете зробити дружньою класу. Неважливо, оголошуєте ви її в public- чи в private-зоні класу. Наприклад:

```
class Anything
{
private:
     int m_value;
public:
     Anything() { m_value = 0; }
     void add(int value) { m_value += value; }

    // Робимо функцію reset() дружньою класу Anything
     friend void reset(Anything &anything);
};

// Функція reset() тепер є другом класу Anything
```

```
void reset(Anything &anything)
{
	// І ми маємо доступ до закритих членів об'єктів класу Anything
	anything.m_value = 0;
}

int main()
{
	Anything one;
	one.add(4); // додаємо 4 до m_value
	reset(one); // скидаємо значення m_value в 0
	return 0;
}
```

Тут ми оголосили функцію reset(), яка приймає об’єкт класу Anything і встановлює m_value значення 0. Оскільки reset() не є членом класу Anything, то в звичайній ситуації функція reset() не мала б доступу до закритих членів Anything. Однак, оскільки ця функція є дружньою класу Anything, вона має доступ до закритих членів Anything.

Необхідно передавати об’єкт Anything в функцію reset() в якості параметра. Це пов’язано з тим, що функція reset() не є методом класу. Вона не має вказівника *this і, крім як передачі об’єкта, вона не зможе взаємодіяти з класом.

*Перевантаження операцій.* Можливість використовувати знаки стандартних операцій для запису виразів як для вбудованих, так і для АТД. У мові C++ для перевантаження операцій використовується ключове слово *operator*, за допомогою якого визначається спеціальна операція-метод (operator function).

Формат операції-методу :

```
тип_поверт_знач operator знак_операції (специф_параметрів)
                    {оператори_тіла_методу}
```

*Перевантаження унарних операцій* Будь-яка унарна операція ⊕ може бути визначена двома способами: або як компонентний метод без параметрів, або як глобальна (можливо дружня) функція з одним параметром. У першому випадку вираз ⊕Z означає виклик Z.*operator* ⊕(), у другому – виклик *operator* ⊕(Z).

Унарні операції, перевантажуванні в рамках певного класу, можуть перевантажуватися тільки через нестатичні компонентні методи без параметрів. Об'єкт класу, що викликає метод, автоматично сприймається як операнд.

Унарні операції, перевантажуванні поза областю класу (як глобальні методу ), повинні мати один параметр типу класу. Передаваний через цей параметр об'єкт сприймається як операнд.

Синтаксис:

1) у першому випадку (опис в області класу):

*тип_поверт_знач operator знак_операції();*

2) в другому випадку (опис поза областю класу):

*тип_поверт_знач operator знак_операції (ідентифікатор_типу)*.

Приклади.

1)

```
class person
{
     int age;
     ...
public:
     void operator++()
     {
          ++age;
     }

};
```

2)

```
class person
{
     int age;
     ...
public:
friend void operator++(person &);

};

void person::operator++(person &ob)
{
     ++ ob.age;
}
```

```
// Використання перевантаженого оператора
void main ()
{
     class person jon;
     ++jon;
}
```

*Перевантаження бінарних операцій.* Будь-яка бінарна операція ⊕ може бути визначена двома способами: або як компонентний метод з одним параметром, або як глобальна (дружня) функція з двома параметрами. У

першому випадку x⊕y означає виклик x.*operator* ⊕(y), у другому – виклик operator ⊕(x, y).

Операції, що перевантажуються всередині класу, можуть перевантажуватися тільки нестатичними компонентними методами з параметрами. Об'єкт класу автоматично сприймається в якості першого операнду.

Операції, що перевантажуються поза областю класу, повинні мати два операнди, один з яких повинен мати тип класу.

Приклади.

```
1).
class person { };
class adresbook
{
     // Містить як компонентні дані множину об'єктів типу
     // person, що представляються як динамічний масив, список, дерево
     person * M;
public:
     person& operator[](int); // доступ до і-го об'єкту
};

person& adresbook::operator[](int i)
{
     return M[i];
}


2).
class person {};
class adresbook
{
// Містить як компонентні дані множину об'єктів типу
// person, що представляються як динамічний масив, список або дерево
...
     person * M;
public:
     // оператор доступу до і-го об'єкту масиву M
     friend person& operator[](const adresbook&, int);
};

person& operator[](const adresbook& ob, int i)
{
     return ob.M[i];

}
```

```
void main ()
{
      adresbook persons;
      person record;
      ...
      // доступ до третього елемента масиву persons.M
      record = persons[3];
}
```

*Перевантаження операції присвоювання*. Операція відрізняється трьома особливостями:

- операція не успадковується;

- операція визначена за замовчуванням для кожного класу в якості операції порозрядного копіювання об'єкта, що стоїть праворуч від знака операції, в об'єкт, що стоїть зліва.

- операція може перевантажуватися тільки в області визначення класу. Це гарантує, що першим операндом завжди буде об’єкт класу (this).

Формат перевантаженої операції присвоювання:

```
назву_класу& operator = (назву_класу &);
```

Відзначимо дві важливі особливості методу *operator=*.

По-перше, в ньому використовується параметр-посилання. Це необхідно для запобігання створення копії об'єкта, який передається через параметр за значенням. У випадку створення копії, вона видаляється викликом деструктора при завершенні роботи методу. Але деструктор звільняє розподілену пам'ять, ще необхідну об'єкту, який є аргументом. Параметр-посилання допомагає вирішити цю проблему.

По-друге, метод *operator=()* повертає не об'єкт, а посилання на нього. Сенс цього той же, що і при використанні параметра-посилання. Метод повертає тимчасовий об'єкт, який видаляється після завершення його роботи. Це означає, що для тимчасової змінної буде викликаний деструктор, який звільняє розподілену пам'ять. Але вона необхідна для присвоювання значення об'єкту. Тому, щоб уникнути створення тимчасового об'єкта, в якості значення, що повертається використовується посилання.

## Порядок виконання роботи.

1. Вибрати клас АТД відповідно до варіантом.

2. Визначити і реалізувати в класі конструктори, деструктор, методи Input (введення з клавіатури) і Print (виведення на екран), перевантажити операцію присвоювання.

3. Написати програму тестування класу і виконати тестування.

4. Доповнити визначення класу заданими перевантаженими операціями (відповідно до варіантом).

5. Реалізувати ці операції. Виконати тестування.

## Методичні вказівки.

1. Класс АТД реалізувати як динамічний масив. Для цього визначення класу повинно мати такі поля:

– вказівник на початок масиву;

– максимальний розмір масиву;

– поточний розмір масиву.

2. Конструктори класу розміщують масив в пам'яті і встановлюють його максимальний і поточний розмір. Для задання максимального розміру масиву використовувати константу, яка визначається поза класом.

3. Щоб у вас не виникало проблем, акуратно працюйте з константними об'єктами. Наприклад:

- конструктор копіювання слід визначити так:

*MyClass (const MyClass& ob);*

- операцію присвоювання перевантажити так:

*MyClass & operator = (const MyClass & ob);*

4. Для зручності реалізації операцій-методів реалізувати в класі private (protected), що працюють безпосередньо з реалізацією класу. Наприклад, для класу множина – це можуть бути наступні методи :

– включити елемент в множину;

– знайти елемент і повернути його індекс;

– видалити елемент;

– визначити, чи належить елемент множині.

Зазначені методи використовуються в реалізації загальнодоступних методів-операцій (operator).

Оператори порівняння завжди перевантажуються в парі (наприклад «>» та «<».

**Зміст звіту.**

1. Титульний лист.

2. Конкретне завдання із зазначенням номера варіанту, реалізованого класу і операцій.

3. Визначення класу.

4. Обґрунтування включення в клас декількох конструкторів, деструктора та операції присвоювання.

5. Пояснити обрання типу пам'яті для об'єктів класу.

6. Реалізація перевантажених операцій з обґрунтуванням обраного способу (метод класу, зовнішня метод, зовнішня дружня функція).

7. Тестові дані та результати тестування.

**Питання для самоконтролю.**

1. Що таке абстрактний тип даних?

2. Наведіть приклади абстрактних типів даних.

3. Які синтаксис/семантика "операції-методу "?

4. Як можна викликати операцію-метод?

5. Чи потрібно перевантажувати операцію присвоювання щодо визначеного користувачем типу даних, наприклад класу? Чому?

6. Чи можна змінити пріоритет перевантаженої операції?

7. Чи можна змінити кількість операндів перевантаженої операції?

8. Чи можна змінити асоціативність перевантаженої операції?

9. Чи можна, використовуючи дружню метод, перевантажити оператор присвоювання?

10. Чи всі оператори мови C ++ можуть бути перевантажені?

11. Якими двома різними способами визначаються перевантажені операції?

12. Чи всі операції можна перевантажити за допомогою глобальної дружної функції ?

13. У яких випадках операцію можна перевантажити тільки глобальною функцією?

14. У яких випадках глобальна операція-функція повинна бути дружньою?

15. Чи обов'язковий у методі *operator* параметр типу "клас" або "посилання на клас"?

16. Чи успадковуються перевантажені операції?

17. Чи можна повторно перевантажити в похідному класі операцію, перевантажену в базовому класі?

18. У чому відмінність синтаксису операції-методу унарної і бінарної операції?

19. Наведіть приклади перевантаження операцій для стандартних типів.

20. Перевантажите операцію "+" для класу "комплексне число".

21. Перевантажите операції "<", ">", "==" для класу "рядок символів

**Додаток.**

Варіанти завдань.

1. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

“+” – додати елемент в множину (set + char);

“+” – об'єднання множин (set + set);

“==” – перевірка множин на рівність (set ==set).

2. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

“–” – видалити елемент з множини (set – char);

“*” – перетин множин (типу set * set);

“>” – порівняння множин.

3. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

“–“ – видалити елемент з множини (set – char);

“>” – перевірка на підмножину (set > subset);

“!=” – перевірка множин на нерівність (set != set).

4. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

“+” – додати елемент в множину (set + char);

“*” – перетин множин (set * set);

int() – потужність множини (приведення до типу int) (int x =(int) set).

5. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

() – конструктор множини на основі масиву (set(*char));

“+” – об'єднання множин (set + set);

“<=” – порівняння множин (set <= set).

6. АТД – множина з елементами типу char. Додатково перевантажити наступні операції:

“==” – перевірка на приналежність (char == set);

“*” – перетин множин (set * set);

“<” – перевірка на підмножину (subset < set);.

7. АТД – однозв’язний список з елементами типу char. Додатково перевантажити наступні операції:

“+” – об'єднати списки (list + list);

“–” – видалити елемент з початку, унарний мінус (–list);

“==” – перевірка на рівність (list == list).

8. АТД – однозв’язний список з елементами типу char. Додатково перевантажити наступні операції:

“+” – додати елемент в початок (char + list);

“+” – додати елемент в початок (char+list);

“==” – перевірка на рівність (list == list).

9. АТД – однозв’язний список з елементами типу char. Додатково перевантажити наступні операції:

“+” – додати елемент в кінець (list + char);

“--” – видалити елемент з кінця (list--);

“!=” – перевірка на нерівність (list != list).

10. АТД – однозв’язний список з елементами типу char. Додатково перевантажити наступні операції:

“[]” – доступ до елементу в заданій позиції, наприклад: int i; char c; list L; c = L[i];

“–” – різниця списків (list – list);

“==” – перевірка на рівність (list == list).

11. АДТ – однозв’язний список з елементами типу char. Додатково перевантажити наступні операції:

“()” – видалити елемент в заданій позиції, наприклад: int i; list L; L(i);

“()” – додати елемент в задану позицію, наприклад: int i; char c; list L; L(c, i);

“!=” – перевірка на нерівність (list != list).

12. АДТ – одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:

“+” – складання векторів (vector – vector, a[i] + b[i] для всіх i);

“[]” – доступ за індексом (vector V; V(3)= double);

“+” – додати число до вектора (vector + double).

13. АТД – одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:

“–“ – віднімання векторів (vector – vector, a[i] – b[i] для всіх i);

“[]” – доступ за індексом (vector V; V(3)= double);

“–“ – відняти від вектора число (vector – double).

14. АТД – одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:

“*” – множення векторів (vector * vector, a[i] * b[i] для всіх i);

“[]” – доступ за індексом (vector V; V(3)= double);

“*” – множення вектора на число (vector * double).

15. АТД – одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:

int () – розмір вектора (приведення до типу int) (int x =(int) vector);

“()” – встановити новий розмір (vector(100));

“–“ – відняти від вектора число (vector – double);

“[]” – доступ за індексом (vector V; V(3)= double).

16. АТД – одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:

“=” – присвоїти всім елементам вектора значення (vector = double);

“[]” – доступ за індексом (vector V; V(3)= double);

“==” – перевірка на рівність (vector == vector).

17. АТД – двовимірний масив (матриця) дійсних чисел. Додатково перевантажити наступні операції:

“[,]” – доступ за індексами (matrix M; M(3,3) = double);

“*” – множення матриць (matrix * matrix);

“*” – множення матриці на число (matrix * double);

18. АТД – двовимірний масив (матриця) дійсних чисел. Додатково перевантажити наступні операції:

“–” – різниця матриць (matrix – matrix);

“–” – відняти з матриці число (matrix – double);;

“==” – перевірка матриць на рівність (matrix == matrix);.

19. АТД – двовимірний масив (матриця) дійсних чисел. Додатково перевантажити наступні операції:

“[]” – (matrix M; M(3,3) = double);

“=” – присвоїти всім елементам матриці значення (matrix = double);

“+” – додавання матриць;

20. АТД – двовимірний масив (матриця) дійсних чисел. Додатково перевантажити наступні операції:

() – доступ по індексу;

== – перевірка матриць на рівність;

++ – транспонувати матрицю (matrix + matrix).