ЛАБОРАТОРНА РОБОТА № 3

ТЕМА: ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Мета

Отримати практичні навики 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.

Короткі теоретичні відомості.

Абстрактний тип даних (АТД).

АТД − тип даних, визначений тільки через операції, які можуть виконуватися над відповідними об'єктами безвідносно до способу представлення цих об'єктів.

АТД включає абстракцію як через параметризацію, так і через специфікацію.

Абстракція через параметризацію може бути здійснена так само, як і для процедур (функцій); використанням параметрів там, де це має сенс.

Абстракція через специфікацію досягається за рахунок того, що операції представляються як частина типу.

Для реалізації АТД потрібно, по-перше, вибрати представлення пам'яті для об'єктів і, по-друге, реалізувати операції в термінах вибраного представлення.

Прикладом абстрактного типу даних є клас в мові С++.

Перевантаження операцій.

Можливість використовувати знаки стандартних операцій для запису виразів як для вбудованих, так і для АТД.

У мові С++ для перевантаження операцій використовується ключове слово **operator**, за допомогою якого визначається спеціальна операція-функція (**operator function**).

Формат операції-функції :

```
тип_возвр_значення operator знак_операції (специф_параметрів)
{оператори_тіла_функції}
```

Перевантаження унарних операцій

- Будь-яка унарна операція ⊕ може бути визначена двома способами: або як компонентна функція без параметрів, або як глобальна (можливо дружня) функція з одним параметром. У першому випадку вираз ⊕ Z означає виклик **Z.operator ⊕ (),** в другому - виклик **operator ⊕(Z).**
- Унарні операції, що перевантажуються у межах певного класу, можуть перевантажуватися тільки через нестатичну компонентну функцію без параметрів. Об'єкт класу, що викликається, автоматично сприймається як операнд.

- Унарні операції, що перевантажуються поза областю класу (як глобальні функції), повинні мати один параметр типу класу. Переданий через цей параметр об'єкт сприймається як операнд.

Синтаксис:

У першому випадку (опис в області класу) :

```
тип_повер_значення operator знак_операції
```

У другому випадку (опис поза областю класу) :

```
тип_повер_значення operator знак_операції(ідентифікатор_типу)
```

Приклади.

1.

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

2.

```
class person
{ int age;
...
public:
…
friend void operator++(person&);
};
void person::operator++(person& ob)
{++ob.age;}
void main()
{
person jon;
++++jon;
}
```

Перевантаження бінарних операцій

- Будь-яка бінарна операція $\oplus$ може бути визначена двома способами: або як компонентна функція з одним параметром, або як глобальна (можливо дружня)

функція з двома параметрами. У першому випадку x⊕y означає виклик **x.operator⊕(y),** в другому - виклик **operator ⊕(x, y).**

- Операції, що перевантажуються всередині класу, можуть перевантажуватися тільки нестатичними компонентними функціями з параметрами. Об'єкт класу, що викликається, автоматично сприймається в якості першого операнда.
- Операції, що перевантажуються поза областю класу, повинні мати два операнди, один з яких повинен мати тип класу.

Приклади.

1)

```
class person{…};
class adresbook
{ // містить в якості компонентних даних безліч об'єктів типу
 // person, що представляються як динамічний масив, список або дерево
…
public:
person& operator[](int);  //доступ до і -го об'єкту
};
person& adresbook:: operator[](int i){. . .}
void main()
{class adresbook persons;
person record;
…
record = persons [3];
}
```

2)

```
class person{.};
class adresbook
{ // містить в якості компонентних даних безліч об'єктів типу
  // person, що представляються як динамічний масив, список або дерево
…
public:
friend person& operator[](const adresbook&, int);
////доступ до і -у об'єкту
};
person& operator[](const adresbook& ob, int i){. . .}
void main(){
adresbook persons;
person record;
…
record = persons [3];
}
```

Перевантаження операції присвоєння

Операція відрізняється трьома особливостями:

- операція не наслідує;
- операція визначена за замовчуванням для кожного класу в якості операції порозрядного копіювання об'єкту, що стоїть праворуч від знаку операції, в об'єкт, що стоїть ліворуч.
- операція може перевантажуватися тільки в області визначення класу. Це гарантує, що першим операндом завжди буде ліводопустимий вираз.

Формат перевантаженої операції присвоєння :

```
ім'я_класу& operator=( ім'я_класу&);
```

Відмітимо дві важливі особливості функції **operator=.**

По-перше, в ній використовується параметр-посилання. Це необхідно для запобігання створенню копії об'єкту, що передається через параметр за значенням. У випадки створення копії, вона видаляється викликом деструктора при завершенні роботи функції. Але деструктор звільняє розподілену пам'ять, ще необхідну об'єкту, який є аргументом. Параметр-посилання допомагає розв'язати цю проблему.

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

Порядок виконання роботи.

1. Вибрати клас АТД відповідно до варіанту.
2. Визначити і реалізувати в класі конструктори, деструктор, функції Input (введення з клавіатури) і Print (висновок на екран), перевантажити операцію присвоєння.
3. Написати програму тестування класу і виконати тестування.
4. Доповнити визначення класу заданими перевантаженими операціями (відповідно до варіанту).
5. Реалізувати ці операції. Виконати тестування.

Методичні вказівки.

1. Клас АТД реалізувати як динамічний масив. Для цього визначення класу повинне мати наступні поля:
    - покажчик 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 завдання максимального масиву використовувати константу, визначену 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 привласнення перевантажити так:
     MyClass& operator = (const MyClass& ob);
4. Для зручності реалізації операцій-функцій реалізувати в класі **private(protected)** - функции, що працюють безпосередньо з реалізацією класу. Наприклад, для класу **множина** це можуть бути наступні функції:
   - включити елемент в множину;
   - знайти елемент і повернути його індекс;
   - видалити елемент;
   - визначити, чи належить елемент множині.

Вказані функції використовуються в реалізації загальнодоступних функцій-операцій (**operator**).

Варіанти завдань.

1. АТД − однонапрямлений список з елементами типу char. Додатково перевантажити наступні операції:
   + + - об'єднати списки (list+list);
   -- − видалити елемент з початку (типу --list);
   = = - перевірка на рівність.
2. АТД − однонапрямлений список з елементами типу char. Додатково перевантажити наступні операції:
   + + − додати елемент в початок(char+list);
   -- − видалити елемент з початку(типу - list);
   = = − перевірка на рівність.
3. АТД − однонапрямлений список з елементами типу char. Додатково перевантажити наступні операції:
   + + − додати елемент в кінець (list+char);
   -- −видалити елемент з кінця (типу list--);
   != − перевірка на нерівність.
4. АТД − однонапрямлений 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];
+ + − об'єднати два списки;
= = − перевірка на рівність.

5. АТД − однонапрямлений 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];
   + + − об'єднати два списки;
   != − перевірка на нерівність.
6. АТД − однонапрямлений 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[з, і];
   != − перевірка на нерівність.
7. АТД − стек. Додатково перевантажити наступні операції:
   + + − додати елемент в стек;
   − − вилучити елемент із стека;
   bool() − перевірка, чи порожній стек.
8. АТД − черга. Додатково перевантажити наступні операції:
   + + − додати елемент;
   − − витягнути елемент;
   bool() - перевірка, чи порожня черга.
9. АТД − одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:
   + + − додавання векторів (a[i]+b[i] для усіх i);
   [] − доступ по індексу;
   + − додати число до вектору (double+vector).
10. АТД − одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:
    - − віднімання векторів (a[i]- b[i] для усіх i);

[] − доступ по індексу;
- − відняти з вектору число (vector - double).

11. АТД − одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:
    * − множення векторів (a[i]*b[i] для усіх i);
    [] − доступ по індексу;
    * − помножити вектор на число (vector*double).
12. АТД − одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:
    int() − розмір вектору;
    () − встановити новий розмір;
    - − відняти з вектору число (vector - double);
    [] − доступ по індексу;
13. АТД − одновимірний масив (вектор) дійсних чисел. Додатково перевантажити наступні операції:
    = − присвоїти усім елементам вектору значення (vector=double);
    [] − доступ по індексу;
    = = − перевірка на рівність;
    != − перевірка на нерівність;
14. АТД − двомірний масив (матриця) дійсних чисел. Додатково перевантажити наступні операції:
    () − доступ по індексу;
    * − множення матриць;
    * − множення матриці на число;
    * − множення числа на матрицю.