ЛЕКЦІЯ 15:

ОСОБЛИВОСТІ МЕХАНІЗМУ ПЕРЕВИЗНАЧЕННЯ ОПЕРАТОРІВ

ЗМІСТ

1. **Механізми перевизначення операторів з використанням функцій-членів класу**
2. **Механізми перевизначення операторів з використанням функцій- не членів класу**
3. **Особливості реалізації оператора присвоєння**
4. **Механізми перевизначення оператора індексації елементів масиву "[]"**
5. **Механізми перевизначення оператора виклику функцій "()"**
6. **Механізми перевизначення рядкових операторів**

ЛІТЕРАТУРА:

1. Horton Ivor, Van Weert Peter Beginning C++20: From Novice to Professional. – Apress, 2020. Chapter 13.
2. Schildt Herbert C++ from the Ground Up. - Third Edition. - McGraw-Hill/Osborne, 2003. Chapter 13.
3. Stroustrup Bjarne The C++ programming language / Bjarne Stroustrup. — Fourth edition. - Pearson Education, Inc, 2013. Chapter 18.

Перевантаження операторів

Оператори перевантажуються за допомогою функції *operator*.

```
тип ім'я_класу::operator#(список_аргументів)
{
  операція_над_класом
}
```

Оператор, що перевантажується, позначається символом "**#**".

тип - тип значення, що повертається заданою операцією, часто збігається з ім'ям класу, для якого перевантажується даний оператор.

список_аргументів визначається кількома факторами.

Операторна функція може бути членом класу або не бути ним. Операторні функції, що не є членами класу, часто визначаються як його "друзі". Операторні функції-члени й функції-не члени класу розрізняються за формою перевантаження.

Обмеження в перевантаженні операторів

1. Майже будь-який існуючий оператор в мові C++ можна перевантажити.

Винятками є:

- тернарний оператор (`?:`);
- оператор sizeof;
- оператор дозволу області видимості (`::`);
- оператор вибору члена (`.`);
- вказівник, як оператор вибору члена (`.*`).

Обмеження в перевантаженні операторів

2. **Можна перевантажити тільки існуючі оператори**. Не можна створювати нові або перейменовувати існуючі. Наприклад, не можна створити оператор ** для виконання операції піднесення до степеня.

3. **Принаймні один з операндів перевантаженого оператора повинен бути користувацького типу даних**.

Не можна перевантажити operator+() для виконання операції додавання значення типу int до значення типу double. Можна перевантажити operator+() для виконання операції додавання значення типу int до об'єкта класу Mystring.

Обмеження в перевантаженні операторів

4. **Початкову кількість операндів, підтримуваних оператором, змінити неможливо**. Тобто з бінарним оператором використовуються тільки два операнди, з унарним — тільки один, з тернарним — тільки три.

5. **Усі оператори зберігають свої пріоритети і асоціативність за замовчуванням** (незалежно від того, для чого вони використовуються), і це не може бути змінено.

Правило: При перевантаженні операторів намагайтеся максимально наближено зберігати функціонал операторів з їх початковими призначеннями.

Механізми перевизначення операторів з використанням функцій-членів класу

Оператори

- присвоювання (=),
- індексу ([]),
- виклику функції (())
- вибору члену (->)

перевантажуються через функції - члени класу — це вимога мови C++.

При перевантаженні через метод класу в якості лівого операнду використовується поточний об’єкт.

```
/* Програма 1.Перевантаження операторів функціями-членами */
#include <iostream>
using namespace std;
class three_d {
    int x, y, z; // 3-вимірні координати
  public:
    three_d() { x = y = z = 0; }
    three_d(int i, int j, int k) { x = i;   y = j;
    z = k; }
   void show();
    three_d operator+(three_d op2);
// Операнд op1 передається неявно.
   three_d operator=(three_d op2);
// Операнд op1 передається неявно.
    three_d operator++(); // Префіксний оператор “++”
    three_d operator++(int notused);
   // Постфіксний оператор “++”
};
```

```
void three_d::show() {
// Відображення координат X, Y, Z.
  cout << x << ", " << y << ", " << z << "\n";
}
three_d three_d::operator+(three_d op2) {
// Перевантаження оп-тора "+”
  three_d temp;
  temp.x = x + op2.x;
  temp.y = y + op2.y;
  temp.z = z + op2.z;
  return temp;
}
three_d three_d::operator=(three_d op2) {
// Перевантаження оп-тора “=“
  x = op2.x;
  y = op2.y;
  z = op2.z;
  return *this;
}
```

```
three_d three_d::operator++() {
// Перевантажений префіксний "++”
  x++; y++; z++; // інкремент координат x, y и z
  return *this;
}

three_d three_d::operator++ (int notused) {
// Постфіксний "++”
  three_d temp = *this; // збереження вихідного значення
  x++;   y++;   z++; // інкремент координат x, y и z
  return temp; // повернення вихідного значення
}
int main() {
  three_d a(1, 2, 3), b(10, 10, 10), c;
  a.show(); b.show();
  c=a+b; c.show(); // додавання об'єктів a й b
  c=a+b+c; c.show(); // додавання об'єктів a, b і c
  c=b=a; c.show(); b.show(); // множинне присвоювання
  ++c;   c.show(); // префіксна форма інкремента
  c++;   c.show(); // постфіксна форма інкремента
  a = ++c; a.show(); c.show(); // a і c однакові
  a = c++; a.show(); c.show(); // Тепер a і c різні
  return 0;
}
```

Перевантаження операторів з використанням функцій членів класу

- Обидві операторні функції мають тільки по одному параметру, незважаючи на те, що вони перевантажують бінарні операції.
  - При перевантаженні бінарного оператора з використанням функції-члена їй передається явно тільки один аргумент. Другий же неявно передається через покажчик this.
  - У рядку

```
temp.x = x + op2.x;
```

під членом **x** мається на увазі член **this->x**, тобто член **x** зв'язується з об'єктом, що викликає дану операторну функцію.

Перевантаження операторів з використанням функцій членів класу

- У всіх випадках неявно передається об'єкт, що вказується ліворуч від символу операції, що став причиною виклику операторної функції.
  - Об'єкт, розташований праворуч від символу операції, передається цій функції як аргумент.
  - У загальному випадку при використанні функції-члена для перевантаження унарного оператора параметри не використовуються взагалі, а для перевантаження бінарного - тільки один параметр.
  - Об'єкт, що викликає операторну функцію, завжди неявно передається через покажчик this.

Перевантаження операторів з використанням функцій членів класу

- ***На відміну від оператора "+", оператор присвоювання приводить до модифікації одного зі своїх аргументів.*** Оскільки функція operator=() викликається об'єктом, що розташований ліворуч від символу присвоювання (=), саме цей об'єкт і модифікується в результаті операції присвоювання.
  - Наприклад, щоб можна було виконувати інструкції

a = b = c = d;

необхідно, щоб операторна функція operator=() повертала об'єкт, що адресується покажчиком this, і щоб цей об'єкт розташовувався ліворуч від оператора "=".

Це дозволить виконати будь-який ланцюжок присвоювань. **Операція присвоювання - це одне з найважливіших застосувань покажчика this.**

Використання функцій-членів для перевантаження унарних операторів

- Операторна функція operator++() інкрементує кожну координату об'єкта й повертає модифікований об'єкт, що цілком узгоджується із традиційною дією оператора "++".
- Прототип постфіксної форми оператора "++" для класу three_d має такий вигляд.
- three_d three_d::operator++(int notused);

- Параметр notused не використовується самою функцією. Він служить індикатором для компілятора, що дозволяє відрізнити префиксну форму оператора інкремента від постфіксної .
  - Функція зберігає поточне значення операнда шляхом виконання такої інструкції:

three_d temp = *this;

Обмеження на перевантаження операторів:

Основна ідея створення перевантаженого оператора - наділити його новими можливостями, які, проте, пов'язані з його первісним призначенням.

- не можна змінювати пріоритет оператора;
- не можна змінювати кількість операндів оператора;
- операторні функції не можуть мати аргументів за замовчуванням;

- **У багатьох випадках порядок проходження операндів має значення.**

Про значення порядку операндів

- У багатьох випадках порядок проходження операндів має значення.
- Наприклад, вираз A+B комутативний, а вираз A-B - ні.
- Реалізуючи перевантажені версії некомутативних операторів, необхідно пам'ятати, який операнд стоїть ліворуч від символу операції, а який - праворуч від нього.
- Наприклад,

```
// Перевантаження оператора віднімання.
three_d three_d::operator-(three_d op2)
{
  three_d temp;
  temp.x = x - op2.x;
  temp.y = y - op2.y;
  temp.z = z - op2.z;
  return temp;
}
```

Перевантаження операторів з використанням функций-не членів класу

Бінарні операторні функції, які не є членами класу, мають два параметри, а унарні (теж не члени) – один (бо не мають доступу до покажчика **this**). Такі функції часто визначаються "друзями" класу.

```
// Перевантаження оператора "+" за допомогою функції-"друга".
#include <iostream>
using namespace std;
class three_d {
    int x, y, z; // 3-вимірні координати
  public:
    three_d() { x = y = z = 0; }
    three_d(int i, int j, int k) { x = i; y = j; z = k; }
    friend three_d operator+(three_d op1, three_d
op2); //Функція-"друг”
    . . .
};
```

Перевантаження операторів з використанням функцій-не членів класу

```
//Реалізація функції-"друга”
three_d operator+(three_d op1, three_d op2)
{
  three_d temp;
  temp.x = op1.x + op2.x;
  temp.y = op1.y + op2.y;
  temp.z = op1.z + op2.z;
  return temp;
}
```

- Функціями-не членами класу не можна перевантажувати оператори: **= () [] ->**

Перевантаження операторів з використанням функцій-не членів класу

- У багатьох випадках при перевантаженні операторів за допомогою функцій-"друзів" немає ніякої переваги в порівнянні з використанням функцій-членів класу.
- Можлива ситуація, у якій функція-"друг" виявляється надзвичайно корисною.
- Наприклад

ob + 10; // буде працювати

- Оскільки об'єкт ob стоїть ліворуч від оператора "+", він викликає перевантажену операторну функцію, що (приблизно) здатна виконати операцію додавання цілочисельного значення з деяким елементом об'єкта ob.

10 + ob; // не буде працювати

- У інструкції об'єкт, розташований ліворуч від оператора "+", являє собою ціле число, тобто значення вбудованого типу, для якого не визначена жодна операція, що включає ціле число й об'єкт класового типу.

Використання функцій-"друзів" дозволяє ставити значення вбудованого типу як праворуч, так і ліворуч від оператора.

```
#include <iostream>
class CL {
    int count;
public:
    void set(int n = 0) {count = n;}
    int get() {return count;}
    CL operator=(CL obj);
    friend CL operator+(CL ob, int i);
    friend CL operator+(int i, CL ob);
};

CL CL::operator=(CL obj) {
    count = obj.count;
    return *this;
}
```

```
// Версія: об'єкт + int-значення.
CL operator+(CL ob, int i) {
    CL temp;
    temp.count = ob.count + i;
    return temp;
}

// Версія: int-значення + об'єкт.
CL operator+(int i, CL ob) {
    CL temp;
    temp.count = ob.count + i;
    return temp;
}

int main() {
    CL o;
    o.set(10);
    std::cout << o.get() << " ";
    o = 10 + o;
    std::cout << o.get() << " ";
    o = o + 12;
    std::cout << o.get();
    return 0;
}
```

```
10 20 32
```

Використання функцій-"друзів" для перевантаження унарних операторів

```
#include <iostream>
using namespace std;
class three_d {
    int x, y, z; // 3-вимірні координати
  public:
    // . . .
    friend three_d operator++(three_d op1);
    // . . .
};
```

Кожна функція-член одержує (у якості неявно переданого) аргумент this, що є покажчиком на об'єкт, що викликав цю функцію.

При перевантаженні унарного оператора за допомогою функції-члена аргументи явно не передаються взагалі.

Єдиним аргументом, необхідним у цій ситуації, є неявний покажчик на викликаючий об'єкт.

Використання функцій-"друзів" для перевантаження унарних операторів

Функції-не члени (у тому числі й "друзі" класу) не одержують покажчик this і, отже, не мають доступу до об'єкта, для якого вони були викликані.

```
/* ЦЕЙ ВАРІАНТ ПРАЦЮВАТИ НЕ БУДЕ бо op1 є
локальним об’єктом */
three_d operator++(three_d op1)
{
  op1.x++;
  op1.y++;
  op1.z++;
  return op1;
// значення виразу сформується правильно,
// а аргумент не зміниться
}
```

```
/* Програма 4. У цій програмі використовуються
перевантажені операторні функції-"друзі" operator++() . */
#include <iostream>
using namespace std;
class three_d {
    int x, y, z; // 3-мірні координати
  public:
    three_d() { x = y = z = 0; }
    three_d(int i, int j, int k) {x = i; y = j; z = k; }
    friend three_d operator+(three_d op1, three_d op2);
    three_d operator=(three_d op2);
    /* Ці функції для перевантаження оператора "++"
використовують  посилальні параметри. */
    friend three_d operator++(three_d &op1);
    friend three_d operator++(three_d &op1, int
notused);
    void show();
};
```

```
// Тепер це функція-"друг".
three_d operator+(three_d op1, three_d op2)
{
  three_d temp;
  temp.x = op1.x + op2.x;
  temp.y = op1.y + op2.y;
  temp.z = op1.z + op2.z;
  return temp;
}

// Перевантаження оператора "=".
three_d three_d::operator=(three_d op2)
{
  x = op2.x;
  y = op2.y;
  z = op2.z;
  return *this;
}
```

```
/* Перевантаження префіксної версії оператора "++" з
використанням функції-"друга". Для цього необхідно
використання посилального параметра. */
 three_d operator++(three_d &op1)
{
  op1.x++;
  op1.y++;
  op1.z++;
  return op1;
}
/* Перевантаження постфіксної  версії оператора "++" з
використанням функції-"друга". Для цього необхідно
використання посилального параметра. */
 three_d operator++(three_d &op1, int notused)
{
  three_d temp = op1;
  op1.x++;
  op1.y++;
  op1.z++;
  return temp;
}
```

```
// Відображення координат X, Y, Z.
void three_d:: show()
{
  cout << x << ", ";
  cout << y << ", ";
  cout << z << "\n";
}

int main(){
  three_d a(1, 2, 3), b(10, 10, 10), c;
  a.show();
  b.show();
  c = a + b; // додавання об'єктів a й b
  c.show();
  c=a+b+c; // додавання об'єктів a, b і з
  c.show();
  c = b = a; // демонстрація множинного присвоювання
  c.show();
  b.show();
  ++c; // префіксна версія інкремента
  c.show();
```

```
c++; // постфіксна версія інкремента
  c. show();
  a = ++c; /* Об'єкт a одержує значення об'єкта c
послідуючим інкрементуванням.*/
  a.show(); // У цьому випадку об'єкти a й c
  c.show(); // мають однакові значення координат.
  a = c++; // Об'єкт a одержує значення об'єкта c до
інкрементування.
  a.show(); // У цьому випадку об'єкти a й c
  c.show(); // мають різні значення координат.
  return 0;
}
```

```
1, 2, 3
10, 10, 10
11, 12, 13
22, 24, 26
1, 2, 3
1, 2, 3
2, 3, 4
3, 4, 5
4, 5, 6
4, 5, 6
4, 5, 6
```

Для реалізації перевантаження операторів варто використовувати функції-члени.

Функції-"друзі" використовуються в C++ в основному для обробки спеціальних ситуацій.

Перевантаження операторів відношення й логічних операторів

Оператори відношення (наприклад, "==" або "<") і логічні оператори (наприклад, "&&" або "||") також можна перевантажувати. Перевантажений оператор відношення або логічний оператор повертає одне із двох можливих значень: true або false. Це відповідає звичайному застосуванню цих операторів і дозволяє використовувати їх в умовних виразах.

```
// Перевантаження оператора "=="
bool three_d::operator==(three_d op2)
{
  if((x == op2.x) && (y == op2.y) && (z == op2.z))
    return true;
  else
    return false;
}
three_d a, b;
  // ...
if(a == b) cout << "a дорівнює b\n";
else cout << "a не дорівнює b\n“;
```

Докладніше про оператор присвоювання:некоректна програма

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
class sample {
    char *s;
  public:
    sample(){s=0;}; // звичайний конструктор
    sample(const sample &ob); // конструктор копії
    ~sample() { if(s) delete [] s; cout << "Звільнення s-
пам'яті.\n"; }
    void show() { cout << s << "\n"; }
    void set(char *str);
   sample operator=(sample &ob);
// перевантажений оператор “=“
};
sample::sample() { // Звичайний конструктор.
  s = new char('\0'); // Член s указує на null-рядок.
}
sample::sample(const sample &ob) { // Конструктор копії.
  s = new char[strlen(ob.s)+1];
  strcpy(s, ob.s);
}
```

```
void sample::set(char *str) { // Завантаження рядка.
  s = new char[strlen(str)+1];
  strcpy(s, str);
}
sample sample::operator=(sample &ob) {
// Перевантаження оператора “=“
  if(strlen(ob.s) > strlen(s)) {
    delete [] s;
    s = new char[strlen(ob.s)+1];  }
  strcpy(s, ob.s);
  return *this;
}
sample input() {
  char instr[80];   sample str;
  cout << "Уведіть рядок: ";  cin >> instr;
  str.set(instr);
  return str;
}
int main() {
  sample ob;
  ob = input();
  ob.show();
  return 0;
}
```

```
Уведіть рядок: Привіт
Звільнення s-пам'яті.
Звільнення s-пам'яті.
*** Тут сміття ***
Звільнення s-пам'яті.
```

Перевантаження оператора індексації масивів ([])

- У C++ (з погляду механізму перевантаження) оператор "[]" уважається бінарним.
- **Його можна перевантажувати тільки для класу й тільки з використанням функції-члена.**
- Загальний формат операторної функції-члена operator[]():

```
тип ім'я_класу::operator[](int індекс)
{
  // ...
}
```

Формально параметр індекс необов'язково повинен мати тип int.

Визначено об'єкт ob, тоді вираз

ob[3]

перетвориться в наступний виклик операторної функції operator[]():

ob.operator[](3)

```
// Перевантаження оператора індексації масивів
#include <iostream>
using namespace std;
const int SIZE = 3;
class atype {
    int a[SIZE];
  public:
    atype() {
      register int i;
      for(i=0; i<SIZE; i++) a[i] = i;
    }
    int &operator[](int i) {return a[i];}
};
int main(){
  atype ob;
  cout << ob[2]; // Відображається число 2.
  cout <<" ";
  ob[2] = 25;
// Оператор "[]" стоїть ліворуч від оператора "=".
  cout << ob[2]; // Тепер відображається число 25.
  return 0;
}
```

```
/* Повернення посилання з операторної функції
operator()[]. */
#include <iostream>
using namespace std;
const int SIZE = 3;
class atype {
    int a[SIZE];
  public:
    atype() {
      register int i;
      for(i=0; i<SIZE; i++) a[i] = i;    }
    int &operator[](int i) {return a[i];}
};
int main(){
  atype ob;
  cout << ob[2]; // Відображається число 2.
  cout <<" ";
  ob[2] = 25;
// Оператор "[]" стоїть ліворуч від оператора "=".
  cout << ob[2]; // Тепер відображається число 25.
  return 0;
}
```

Перевантаження оператора індексації масивів ([])

- Одна з переваг перевантаження оператора "[]" полягає в тому, що з його допомогою ми можемо забезпечити засіб реалізації безпечної індексації масивів.
- Якщо створити клас, що містить масив, і дозволити доступ до цього масиву тільки через перевантажений оператор індексації "[]", то можливе перехоплення індексу, значення якого вийшло за дозволені межі.

```
// Контроль виходу за припустимий інтервал індексів класу atype
#include <iostream>
#include <cstdlib>
#include <windows.h>
using namespace std;
const int SIZE_ = 3;
class atype {
    int a[SIZE_];
  public:
    atype() { register int i; for(i=0; i<SIZE_; i++) a[i] = i;
}
    int &operator[](int i); };
int &atype::operator[](int i) {
  if (i < 0 || i > SIZE_ - 1) {
    cout << "\n Значення індексу "<< i << " виходить за межі
масиву\n";
    exit(1);  }
  return a[i];}
int main() {
 SetConsoleCP(1251);
 SetConsoleOutputCP(1251);
  atype ob;
  cout << ob[3]; // Генерується помилка часу виконання
}
```

Перевантаження оператора "()"

- При його перевантаженні створюється не новий спосіб виклику функцій, а операторна функція, якій можна передати довільне число параметрів.
- Припустимо, що деякий клас містить наступне оголошення перевантаженої операторної функції.

```
int operator()(float f, char *p);
```

- Якщо в програмі створюється об'єкт ob цього класу, то інструкція

```
ob (99.57, "перевантаження");
```

перетвориться в наступний виклик операторної функції operator():

```
operator() (99.57, "перевантаження");
```

Перевантаження оператора "()"

- У загальному випадку при перевантаженні оператора "()" визначаються параметри, які необхідно передати функції **operator**().
- При використанні оператора "()" у програмі аргументи, що задають при цьому, копіюються в ці параметри.
- Об'єкт, що генерує виклик операторної функції адресується покажчиком **this**.

```
#include <iostream>
// Демонстрація перевантаження оператора "()".
using namespace std;
class three_d {
    int x, y, z; // 3-вимірні координати
  public:
    three_d() { x = y = z = 0; }
    three_d(int i, int j, int k) {x = i; y = j; z = k; }
    three_d operator()(int a, int b, int c);
    void show();
};
three_d three_d::operator()(int a,int b,int c) {
 // Перевантаження "()"
  three_d temp;
  temp.x = x + a;  temp.y = y + b;  temp.z = z + c;
  return temp;}
void three_d::show() {   // Відображення координат x, y, z.
  cout << x << ", " << y << ", " << z << "\n";
}
int main() {
  three_d ob1(1, 2, 3), ob2;
  ob2 = ob1(10, 11, 12); // виклик функції operator()
  cout << "ob1: ";  ob1.show();
  cout << "ob2: ";  ob2.show();
  return 0;}
```

```
ob1: 1, 2, 3
ob2: 11, 13, 15
```

Перевантаження інших операторів

- За винятком таких операторів, як new, delete, ->, ->* і "кома", інші C++-оператори можна перевантажувати таким же способом, що був показаний у попередніх прикладах.

```
#include <iostream>
#include <cstring>
using namespace std;
class str_type {
// Створення "класового“ рядкового типу str_type.
    char string[80];
  public:
    str_type(char *str = "") { strcpy(string, str); }
    str_type operator+(str_type str);
// конкатенація рядків
    str_type operator+(char *str);
    str_type operator=(str_type str);
// присвоювання рядків
    str_type operator=(char *str);
    void show_str() { cout << string; }
// Виведення рядка
};
// Конкатенація 2-х рядків
str_type str_type::operator+(str_type str) {
  str_type temp;
  strcpy(temp.string, string);
  strcat(temp.string, str.string);
  return temp;
}
```

```
// Присвоювання одного рядка іншому
str_type str_type::operator=(str_type str) {
  strcpy(string, str.string);
  return *this;
}

int main() {
  str_type a("Всім ");
  str_type b("привіт ");
  str_type c;
  c = a + b;
  c.show_str();
  return 0;
}
```

Всімпривіт

```
// Удосконалення рядкового класу.
#include <iostream>
#include <cstring>
using namespace std;
class str_type {
    char string[80];
  public:
    str_type(char *str = "") { strcpy(string, str); }
    str_type operator+(str_type str);
    str_type operator+(char *str);
    str_type operator=(str_type str);
    str_type operator=(char *str);
    void show_str() { cout << string; }
};
str_type str_type::operator+(str_type str) {
  str_type temp;
  strcpy(temp.string, string);
  strcat(temp.string, str.string);
  return temp;
}
str_type str_type::operator=(str_type str) {
  strcpy(string, str.string);
  return *this;
}
```

```
str_type str_type::operator=(char *str) {
  str_type temp;
  strcpy(string, str);
  strcpy(temp.string, string);
  return temp;
}
str_type str_type::operator+(char *str) {
  str_type temp;
  strcpy(temp.string, string);
  strcat(temp.string, str);
  return temp;
}

int main() {
  str_type a("Привіт "), b("всім"), c;
  c = a + b;   c.show_str();   cout << "\n";
  a = "для програмування, тому що ";  a.show_str();
  cout << "\n";
  b = c = "C++ це супер";
  c = c + " " + a + " " + b;  c.show_str();
  return 0;
}
```

```
Привіт всім
для програмування, тому що
C++ це супер для програмування, тому що C++ це супер
```