Лекція 12.
Особливості побудови класів з полями-вказівниками.

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

План на сьогодні

1. Приклад класу Vector
2. Методи копіювання та вводу
3. Оператор присвоєння
4. Деструктор
5. Оператори введення та виведення
6. Ідіома RAII

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад класу Vector. Управління ресурсами та методи.

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад класу Vector

- Клас **Vector** містить поля: size (кількість елементів) і array (динамічний масив елементів типу int).
- Оскільки клас містить поле-вказівник, він повинен мати **конструктор копіювання, оператор присвоєння** та **деструктор** для управління ресурсами.
  Правило трьох («Закон великої трійки»)

```
class Vector {
private:
    int* array;
    size_t size;
public:
    Vector() : array(0), size(0) {}
    Vector(size_t size, int val = 0);
    Vector(int* a, size_t s) : size(s) {
        array = new int[size];
        for (int i = 0; i < size; i++)
            array[i] = a[i];
    }
    Vector(const Vector &);
    void input();
    Vector& operator=(const Vector& v);
    int& operator[](size_t pos);
    ~Vector();
};
```

Конструктор копіювання

Оператор присвоєння

Деструктор

Методи копіювання та вводу

Конструктор копіювання

```
Vector::Vector(const Vector& v) {
    size = v.size;
    array = new int[size];
    for (size_t i = 0; i < size; ++i)
        array[i] = v.array[i];
}
```

Введення даних

```
void Vector::input() {
cout <<“Input ”<<x.size<<“ elements” << endl;
   for (size_t i = 0; i < x.size; ++i)
       is >> x.array[i];
    return is;
}
```

Приклад використання

```
void main() {
    size_t n;
    cout <<“n=”;
    cin>>n;
    Vector v1(n);
    v1.input();
    int arr[3] = {1, 2, 3};
    Vector v2(arr, 3);
    Vector v3(v1);
}
```

Виклик методу введення даних

Виклик конструктора з параметрами

Виклик конструктора копіювання

Оператор присвоєння

Оператор присвоєння

```
Vector& Vector::operator=(const Vector& v) {
    if (this != &v) {
        if (array != 0) delete[] array;
        array = new int[size = v.size];
        for (int i = 0; i < size; ++i)
            array[i] = v.array[i];
    }
    return *this;
}
```

Приклад використання

```
void main() {
    Vector v1;
    v1.input();
    int arr[3] = {1, 2, 3};
    Vector v2(arr, 3);
    Vector v3(v1);
    v2 = v1;
}
```

Виклик оператора присвоєння

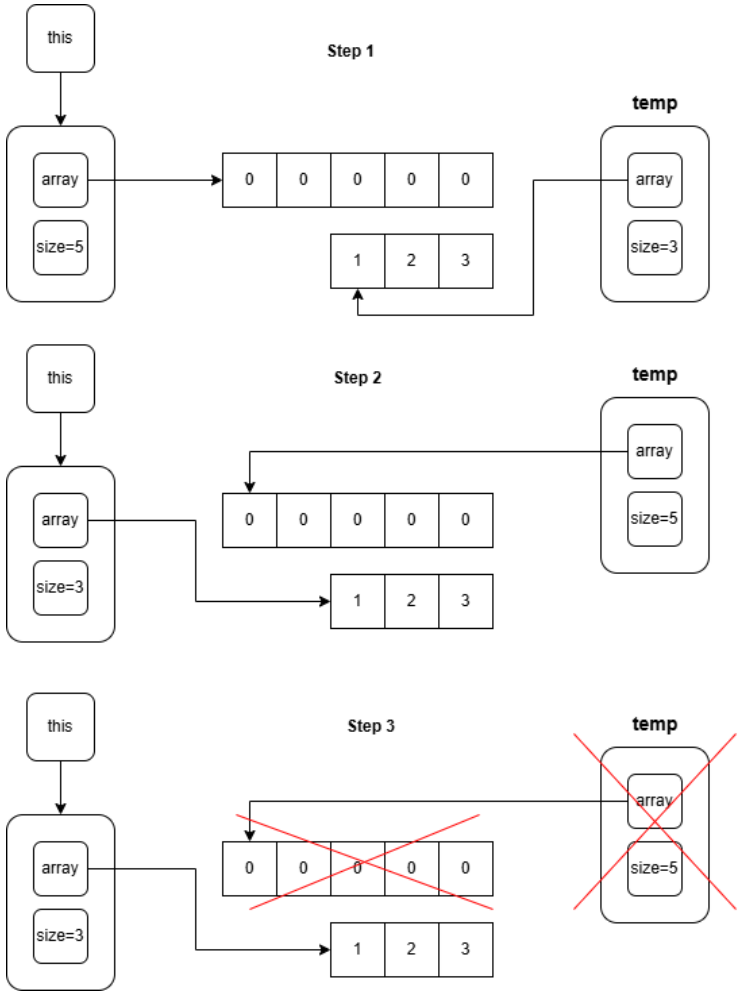
Copy-and-Swap Idiom

```
void swap(Vector& vLeft, Vector& vRight)
{
    std::swap(vLeft.size, vRight.size);
    std::swap(vLeft.array, vRight.array);
}
```

```
Vector& Vector::operator = (const Vector& v)
{
    if (this != &v) {
        Vector temp(v);
        swap(*this, temp);
    }
    return *this;
}
```

```
{
    Vector z(5);
    std::cout << "Vector z:\n";
    std::cout << z;
    int arr[3] = { 1,2,3 };
    Vector y(arr, 3);
    std::cout << "Vector y:\n";
    std::cout << y;
    z = y;
    std::cout << "Vector z:\n";
    std::cout << z;
}
```

```
Vector z:
0 0 0 0 0
Vector y:
1 2 3
Vector z:
1 2 3
```



Деструктор

- **Призначення**: Деструктор виконує очищення ресурсів, захоплених об'єктом протягом його життєвого циклу.
- **Особливості**: Подібно до конструкторів і operator=, деструктор визначається розробником для належного керування пам'яттю.
- **Правило трьох**: Якщо клас потребує деструктора, зазвичай також необхідно визначити конструктор копіювання та оператор присвоєння.

```
Vector::~Vector() {
    if (array != nullptr) delete[] array;
}
```

Оператори введення та виведення

```
class Vector {
private:
    int* array;
    size_t size;
public:
    friend istream& operator >> (istream& is, Vector& x);
    friend ostream& operator << (ostream& os, const Vector& x);
};
```

```
void main() {
    size_t n;
    cout << “n=”;
    cin >> n;
    Vector v1(n), v2(n);
    cin >> v1 >> v2;
    cout << "vector1:\n" << v1;
    cout << "vector2:\n" << v2;
}
```

```
istream& operator >> (istream& is, Vector& x) {
   cout <<“Input ”<<x.size<<“ elements” << endl;
   for (int i = 0; i < x.size; i++)
       is >> x.array[i];
    return is;
}
```

```
ostream& operator << (ostream& os, const Vector& x) {
    os << "\nVector size = " << x.size << endl;
    os << "Vector elements: ";
    for (int i = 0; i < x.size; i++)
        os << x.array[i] << ' ';
    return os;
}
```

Оператор індексу

```
int& Vector::operator[](size_t pos)
{
  if (pos < size) {
    return array[pos];
  }
  cout << ”error: out of range”;
}
```

- Якщо внутрішньою структурою є масив, то найпростіший індекс типу **size_t**:

  константний селектор – щоб отримати ("зчитати") дані, результат **r-value**

  ```
  const X& operator[](size_t)const;
  ```

  неконстантний модифікатором– щоб задати ("записати") дані, результат (**r-value**)

  ```
  X& operator[](size_t);
  ```

- Якщо внутрішня структура сформована з пар об'єктів типів Y і X, то індексом може бути тип Y

  ```
  const X& operator[](const Y&)const;
        X& operator[](const Y&);
  ```

Ідіома RAII (Resource Acquisition Is Initialization)

- **RAII**: отримання ресурсу прив'язується до часу життя об'єкта.
- **Принцип роботи**: ресурс отримується в конструкторі і звільняється в деструкторі.
- **Перевага**: автоматичне звільнення ресурсів при виході об'єкта з області видимості, запобігаючи витокам пам'яті.
- **Приклади**: std::string, std::vector у стандартній бібліотеці C++.

```
class Vector {
private:
    int* array;
    size_t size;
public:
    // Конструктор виділяє пам'ять
    Vector(size_t s) : size(s), array(new int[s]) {}
    // Деструктор звільняє пам'ять
    ~Vector() {
        delete[ ] array;
    }
};
```

Дякую!

FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY