

ЛАБОРАТОРНА РОБОТА №16. РОБОТА З ДИНАМІЧНОЮ ПАМ'ЯТТЮ

Тема. Системна робота з динамічною пам'яттю.

Мета. Дослідити особливості мови C++ при роботі з динамічною пам'яттю.

1 ВИМОГИ

1.1 Розробник

- Котенко Сергій Миколайович;
- Студент групи КІТ 102.8(а);
- 12-06-2019р..

1.2 Загальне завдання

Загальне завдання.

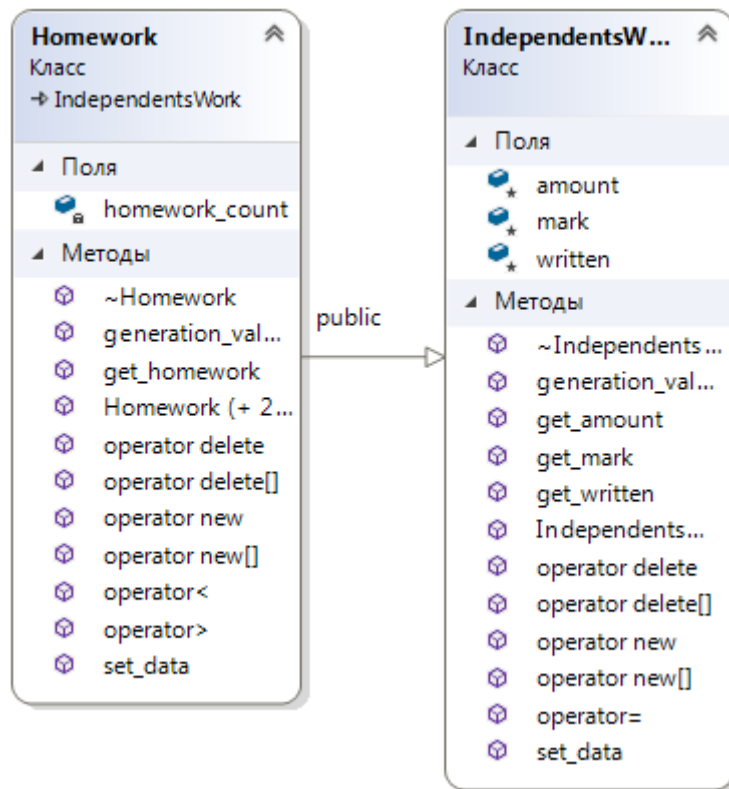
Маючи класи з прикладної області РЗ (тільки базовий клас та клас/класи спадкоємці), перевантажити оператори `new` / `new []` та `delete` / `delete []`. Продемонструвати їх роботу і роботу операторів розміщення `new` / `delete` при розробці власного менеджера пам'яті (сховища). Детальна інформація про власне сховище: є статично виділений масив заданого обсягу. Організувати виділення і звільнення пам'яті елементів ієрархії класів тільки в рамках даного сховища.

Додаткові умови виконання завдання:

- продемонструвати відсутність витоків пам'яті

2 ОПИС ПРОГРАМИ

2.1 Опис логічної структури



Діаграма класу IndependentWork:

- ✓ `~IndependentWork` - Деструктор класу;
- ✓ `generation_values` – Генерація випадкових значень;
- ✓ `get_amount`, `get_mark`, `get_written` - Отримання даних;
- ✓ `IndependentWork` - Конструктор класу;
- ✓ `operator delete`, `operator delete[]` , `operator new`, `operator new[]` , `operator=` - Перевантаження операторів;
- ✓ `set_data` - Встановлення значень .

Діаграма класу Homework:

- ✓ `~Homework` - Деструктор класу;
- ✓ `generation_values` – Генерація випадкових значень;
- ✓ `get_homework` - Отримання даних;
- ✓ `Homework` - Конструктор класу;
- ✓ `operator delete`, `operator delete[]` , `operator new`, `operator new[]` , `operator=` , `operator<`, `operator>` - Перевантаження операторів;
- ✓ `set_data` - Встановлення значень .

2.2 Фрагменти коду

```
void* IndependentsWork::operator new(size_t size) {
    void* pointer = malloc(size);
    if (pointer == nullptr) {
        throw std::bad_alloc();
    }
    std::cout << "Memory was allocated for " << size << " elements" << std::endl;
    return pointer;
}

void IndependentsWork::operator delete(void *pointer) {
    free(pointer);
    std::cout << "The memory has been freed" << std::endl;
}
```

Рисунок 2.1 – Перевантаження операторів new та delete

```
void* IndependentsWork::operator new[](size_t size) {
    void* pointer = malloc(size);
    if (pointer == nullptr) {
        throw std::bad_alloc();
    }
    std::cout << "Memory was allocated for " << size << " elements" << std::endl;
    return pointer;
}

void IndependentsWork::operator delete[](void *pointer) {
    free(pointer);
    std::cout << "The memory has been freed" << std::endl;
}
```

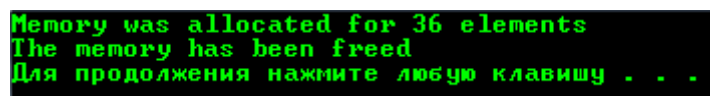
Рисунок 2.2 – Перевантаження операторів new[] та delete[]

```
IndependentsWork* work_or_not = new Homework[2];
delete[] work_or_not;
```

Рисунок 2.3 – Перевірка працездатності

3 ВАРІАНТИ ВИКОРИСТАННЯ

3.1 Ілюстрація роботи програми



```
Memory was allocated for 36 elements
The memory has been freed
Для продовження натисніть будь-яку клавішу . . .
```

Рисунок 3.1 – Результат виділення та очищення пам'яті

ВИСНОВОК

В інтегрованому середовищі *Visual Studio* розроблена програма мовою C++. Виконання програми дозволяє продемонструвати коректність роботи перевантаження операторів new / new [] та delete / delete [].