

**ЛАБОРАТОРНА РОБОТА № 7**  
**Черга та черга з пріоритетом**  
з курсу “Алгоритми та структури даних”

Виконав:

Студент групи ПМІ-16

Бевз Маркіян Юрійович

**Мета:** Ознайомитись з чергою та чергою з пріоритетом, розробити клас черги та черги з пріоритетом, який дозволяє виконувати операції додавання, видалення та перевірки наявності елементів у звичайній черзі та черзі з пріоритетом.

### Принцип роботи черги:

- **Додавання елементу (Enqueue):** Новий елемент додається до кінця черги.
- **Видалення елементу (Dequeue):** Перший елемент черги видаляється.
- **Перевірка наявності елементу (IsEmpty):** Перевіряється, чи є черга порожньою.
- **Отримання першого елементу (Front):** Повертається перший елемент черги без його видалення.
- **Отримання розміру черги (GetSize):** Повертається розмір черги.

**Хід роботи:** Після вивчення принципу роботи черги я реалізував клас Queue на основі двозв'язного списку, який містить методи для додавання, знаходження розміру черги, видалення та перевірки наявності елементів у черзі. Також було створено 8 тестів для перевірки правильності роботи цих методів. Нижче буде прикріплено результат виконання програми, тестів, та їх код.

```
Queue size: 3
First elem in queue: 10
dequeue :10
First new elem in queue: 20
New queue size: 2
```

▲ ✓ Queue tests (16)	3 ms
▲ ✓ QueueUnitTest .	< 1 ms
▲ ✓ QueueTests (..	< 1 ms
✓ TestMultip...	< 1 ms
✓ TestGetFro...	< 1 ms
✓ TestGetFro...	< 1 ms
✓ TestEnque...	< 1 ms
✓ TestEnque...	< 1 ms
✓ TestEnque...	< 1 ms
✓ TestEmpty...	< 1 ms
✓ TestDeque...	< 1 ms

```

TEST_CLASS(QueueTests)
{
public:

    TEST_METHOD(TestEnqueue)
    {
        Queue queue;
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);

        Assert::AreEqual(3, queue.getSize());
    }

    TEST_METHOD(TestEnqueueDequeue)
    {
        Queue queue;
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);

        queue.dequeue();

        Assert::AreEqual(2, queue.getSize());
    }

    TEST_METHOD(TestDequeueFromEmptyQueue)
    {
        Queue queue;
        try {
            queue.dequeue();
            Assert::Fail(L"Expected an exception");
        }
        catch (const std::exception&) {
            // Test passed: Expected an exception
        }
    }

    TEST_METHOD(TestGetFront)
    {
        Queue queue;
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);

        Assert::AreEqual(1, queue.getFront());
    }
}

```

TEST\_METHOD(TestGetFrontFromEmptyQueue)

```
Queue queue;
try {
    queue.getFront();
    Assert::Fail(L"Expected an exception");
}
catch (const std::exception&) {
    // Test passed: Expected an exception
}
```

TEST\_METHOD(TestMultipleEnqueueDequeue)

```
Queue queue;
queue.enqueue(1);
queue.enqueue(2);
queue.dequeue();
queue.enqueue(3);
queue.dequeue();

Assert::AreEqual(1, queue.getSize());
```

TEST\_METHOD(TestEnqueueDifferentDataTypes)

```
Queue queue;
queue.enqueue(1);
queue.enqueue(2.5);
queue.enqueue('c');

Assert::AreEqual(3, queue.getSize());
```

TEST\_METHOD(TestEmptyQueueAfterDequeueAll)

```
Queue queue;
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);
queue.dequeue();
queue.dequeue();
queue.dequeue();

Assert::IsTrue(queue.isEmpty());
```

## Принцип роботи черги з пріоритетом:

- **Додавання елементу з пріоритетом (Enqueue):** Новий елемент додається до черги з урахуванням його пріоритету.
- **Видалення елементу (Dequeue):** Перший елемент черги, який має найвищий пріоритет, видаляється.
- **Перевірка наявності елементу (IsEmpty):** Перевіряється, чи є черга з пріоритетом порожньою.
- **Отримання першого елементу (Front):** Повертається перший елемент черги з найвищим пріоритетом без його видалення.
- **Отримання розміру черги (GetSize):** Повертається розмір черги пріоритетом.

**Хід роботи:** Після вивчення принципу роботи черги з пріоритетом я реалізував клас `PriorityQueue` на основі однозв'язного списку, який містить методи для додавання, знаходження розміру черги, видалення та перевірки наявності елементів у черзі з пріоритетом. Також було створено 8 тестів для перевірки правильності роботи цих методів. Нижче буде прикріплено результат виконання програми, тестів, та їх код.

```
Priority queue:  
Queue size: 3  
First elem in queue: 2  
First new elem in queue: 11  
New queue size: 2
```

▲	✓	PriorityQueue...	< 1 ms
▲	✓	PriorityQueu...	< 1 ms
	✓	TestMultip...	< 1 ms
	✓	TestGetFro...	< 1 ms
	✓	TestGetFro...	< 1 ms
	✓	TestEnque...	< 1 ms
	✓	TestEnque...	< 1 ms
	✓	TestEnque...	< 1 ms
	✓	TestEnque...	< 1 ms
	✓	TestDeque...	< 1 ms

```

TEST_CLASS(PriorityQueueTests)
{
public:

    TEST_METHOD(TestEnqueueWithPriority)
    {
        PriorityQueue queue;
        queue.Enqueue(1, 2); // Элемент с приоритетом 2
        queue.Enqueue(2, 1); // Элемент с приоритетом 1
        queue.Enqueue(3, 3); // Элемент с приоритетом 3

        Assert::AreEqual(3, queue.GetSize());
    }

    TEST_METHOD(TestEnqueueDequeueWithPriority)
    {
        PriorityQueue queue;
        queue.Enqueue(1, 2);
        queue.Enqueue(2, 1);
        queue.Enqueue(3, 3);

        queue.dequeue();

        Assert::AreEqual(2, queue.GetSize());
    }

    TEST_METHOD(TestDequeueFromEmptyPriorityQueue)
    {
        PriorityQueue queue;
        try {
            queue.dequeue();
            Assert::Fail(L"Expected an exception");
        }
        catch (const std::exception&) {
            // Test passed: Expected an exception
        }
    }

    TEST_METHOD(TestGetFrontWithPriority)
    {
        PriorityQueue queue;
        queue.Enqueue(1, 2);
        queue.Enqueue(2, 1);
        queue.Enqueue(3, 3);

        Assert::AreEqual(2, queue.getFront());
    }
}

```

No issues found

```

TEST_METHOD(TestGetFrontFromEmptyPriorityQueue)
{
    PriorityQueue queue;
    try {
        queue.getFront();
        Assert::Fail(L"Expected an exception");
    }
    catch (const std::exception&) {
        // Test passed: Expected an exception
    }
}

TEST_METHOD(TestMultipleEnqueueDequeueWithPriority)
{
    PriorityQueue queue;
    queue.Enqueue(1, 2);
    queue.Enqueue(2, 1);
    queue.dequeue();
    queue.Enqueue(3, 3);
    queue.dequeue();

    Assert::AreEqual(1, queue.getSize());
}

TEST_METHOD(TestEnqueueWithSamePriority)
{
    PriorityQueue queue;
    queue.Enqueue(1, 2);
    queue.Enqueue(2, 2);
    queue.Enqueue(3, 2);

    Assert::AreEqual(3, queue.getSize());
}

TEST_METHOD(TestEnqueueDifferentDataTypesWithPriority)
{
    PriorityQueue queue;
    queue.Enqueue(1, 2);
    queue.Enqueue(2.5, 1);
    queue.Enqueue('c', 3);

    Assert::AreEqual(3, queue.getSize());
}

```

**Висновок:** Я ознайомився з чергою та чергою з пріоритетом, розробив клас черги та черги з пріоритетом, який дозволяє виконувати операції додавання, видалення та перевірки наявності елементів у звичайній черзі та черзі з пріоритетом.