Курсові роботи з курсу “Об’єктно орієтоване програмування”

Загальні вимоги:

- Обов’язкова наявність конструкторів по замовченню, копіювання, переміщення та деструктора.
- Методи класів не повинні бути “прив’язаними” до конкретних потоків даних або файлів.
- Наявність ітераторів для класів-контейнерів є обов'язковою умовою. Ітератори мають бути реалізовані як вкладений клас, із підтримкою стандартних операцій. Передбачити обов’язкові методи та операції:
  - Метод **begin()** - повертає ітератор на початок.
  - Метод **end()** - повертає ітератор на кінець.
  - Операції інкременту (**++**), декременту (--), розіменування (*****), порівняння (**==**, **!=**).
- Основні вимоги до класів-контейнерів:
  - Дінамічно зберігати елементи.
  - Автоматично розширювати розмір контейнеру.
  - Необхідно перевантажити основні оператори: рівності (**==**), присвоєння (**=**), інкременту (**++**) , декременту (**--**), індексу (**[]**).
  - Базові методи:
    - Додавання нового елементу.
    - Вилучення елементів по індексу, ітератору та по його значенню.
    - Отримати розмір контейнеру (кількість елементів).
    - Модіфікація значення вже існуючого елементу.
    - Пошук по значенню з повертанням ітератора.
    - Підтримка компараторів та/або унарних предикатів у функціях сортування та пошуку (це дозволить користувачам визначати власні умови порівняння або фільтрації під час виконання операцій).
- Реалізація методів має бути винесена за межі опису класу.
- Усі методи повині бути прокометовані: призначення, та коротко опис алгоритму, який реалізован у методі.

- Розробити основну програму, яка демонструє використання розробленного класу.
- Розробити не меньше трех тестів для перевірки роботи програми. Описати які особливості роботи програми перевіряє той чи іншій тест (де це доречно).

Теми робіт

1. Масив з автоматичним розширенням (Dynamic Array / **Vector**). Аналог **std::vector**, шаблон класу **Vector<T>**, що сам розширюється при досягненні ліміту.
2. Файл, як масив з автоматичним розширенням (**FileVector**). Шаблон класу, який надає інтерфейс для роботи з типизованим файлом, як з масивом.
3. Універсальний стек (**Stack**). Шаблон класу **Stack<T>**, що реалізує стек з базовими операціями: **push, pop, top, isEmpty**.
4. Черга (**Queue**). Шаблон класу **Queue<T>** з методами **enqueue, dequeue, peek.**
5. Однозв’язний список (**Linked1List**). Шаблон класу **Linked1List<T>.**
6. Двозв’язний список (**Linked2List**). Шаблон класу **Linked2List<T>.**
7. Калькулятор (**Calculator**). Шаблон класу для базових арифметичних операцій з різними типами чисел. Конструктор класу отримує рядок з арифметичним виразом, наприклад **5.6+4*(7-67.7)/56**. Кількість відкритих дужок не обмежана. У випадку некоректно заданого арифметичного виразу повідомляє про помилку.
8. Універсальний словник (**Map / Dictionary**). Шаблон класу **Map<KeyType, ValueType>** на основі хеш-таблиці або бінарного дерева.
9. Матриця (**Matrix**). Шаблон класу для двовимірної матриці з базовими операціями (додавання, множення, транспонування, визначник).
10. Довгі цілі числа (**BigInteger**). Шаблон класу для роботи з довгими цілими числами. Передбачити операції додавання та віднімання. Примітка: Довге ціле число – це ціле число, значення якого не може бути записано ні в жодний

стандартний тип даних. Клас повинен зберігати одне таке ціле число, виконувати на ним математичні дії такі як додавання та віднімання іншого довгого числа або короткого.

11. Сортувальник (**Sorter**). Шаблон з реалізацією різних алгоритмів сортування (**Bubble**, **Quick**, **Merge** тощо). Шаблон сортування з параметром політики сортування (за не зростанням, не спаданням).

12. Символьний рядок (**String**). Аналог класу **string**. Повинен підтримувати операції пошуку, реверсу, видалення, вставки. Додавання та видалення повинно супроводжуватись дінамічним перерозподілом пам’яті. Клас **string** стандартної бібліотеки використовувати заборонено.

13. Контейнер **Points**. Кожен елемент такого контейнеру має такі властивості: координата **x** (ціле значення), координата **y** (ціле значення), віддаленність точки від початку системи координат, колір точки, її порядковий номер. Контейнер підтримує операції додавання, видалення, пошук за вказаним параметром, сорування по вказаному параметру, пошук ареалів заданого кольору (кількість та площу), визначення мінімального прямокутника, в межах якого існують точки (його площу, та координати вершин такого прямокутника).

14. Розробити шаблон класу-контейнер **Students<T>**, який зберігає об'єкти довільного типу **T**. Контейнер повинен підтримувати такі операції: додавання, вставку, видалення (за індексом та/або ітератором), пошук (з поверненням індексу та/або ітератора), сортування, доступ до елементів (**get** / **set**). Структура типу **T** (наприклад, **Student**) визначається в основній програмі.

15. Множина (**Set**). Розробити шаблон класів для роботи з множинами. Передбачити методи додавання, видалення, пошуку. Навігація по контейнеру за допомогою ітераторів.

16. Комплексне число (**Complex**). Розробити шаблон класів для роботи з комплексними числами. Передбачити основні операції над комплексними числами (Арифметика комплексних чисел).

17. **Календар**. Розробити клас для роботи з календарними датами. Передбачити можливість отримання дати через задану кулькість секунд, додавання/віднімання днів, перевірка на високосний рік, обчислення різниці між датами.

18. Клас “**Вектор у 2D/3D**”. Операції над векторами: довжина, скалярний добуток, векторний добуток, кут між векторами.

19. Клас “**Поліном**”. Реалізувати клас для роботи з многочленами (додавання, множення, похідна, значення в точці).

20. Клас “**Интеграл**”. Реалізувати клас, який знаходить числове значення інтегралу заданої функції на заданому інтервалі.

21. Клас “**Багаторозрядна арифметика**”. Реалізувати клас для роботи з багаторозрядними числами. Передбачити основні арифметичні операції.

22. Клас “**Геометричні фігури**”. Абстрактний клас Shape, похідні: Circle, Rectangle, Triangle. Методи: обчислення площі, периметра, масштабування.

23. Клас “**Файлова система**” (імітація структури файлів). Створити класи File, Directory, FileSystem, де можна створювати, видаляти, шукати файли/папки.

24. Клас “**Текстовий редактор**”. Клас, який може зберігати текст, додавати/видаляти символи (редагувати), рахувати слова, абзаци, символи. Клас повинен підтримувати багаторядковий текст. Передбачити методи збереження тексту у файлі та його відновлення з файлу у пам’ять. Розмір тексту є динамічним.

25. Розробка класу **electric_field** для розрахунку векторного поля, створеного двома електричними зарядами. Клас формує двовимірну сітку точок, у яких розраховується значення та напрям вектора напруженості електричного поля (двовимірний масив, елементи якого є пара значень: модуль та напрям вектора напруженості електричного поля). При зміні координат зарядів, величин зарядів та їх знаків значення у вузлах сітки перераховуються. Розмір сітки визначається при створені екземпляру класу **field**.

26. Моделювання руху планет у спрощеній гравітаційній системі засобами об’єктно-орієнтованого програмування. Розробити програму, що моделює рух двох небесних тіл під дією гравітаційного притягання, використовуючи принципи об’єктно-орієнтованого програмування. Візуалізація з використанням бібліотеки QT.

27. Гра “**Pac-man**”. Створити програмне забезпечення — спрощену версію гри *Pac-Man*, реалізовану з використанням принципів об’єктно-орієнтованого програмування, з чітким поділом відповідальностей між класами, підтримкою базової ігрової логіки та взаємодії користувача з грою. Реалізувати класи, що відповідають за:

    - головного героя (Pac-Man),
    - привидів (вороги),
    - карту/лабіринт,
    - їжу (пункти),
    - ігровий процес (логіку гри, рахунок, стан гри).

    Усі класи персонажів є похідними від єдиного абстрактного класу.

Застереження: У курсовій роботі забороняється використовувати будь яки засоби бібліотеки **STL** (**Standard Template Library**).

Додаток

Приклад оформлення тексту класу:

```
#include <iostream>
#include <utility>                  // Для std::move

template <typename T>
class Box {
  private:
    T value;

  public:
    // Конструктори
    Box();                          // Конструктор за замовчуванням
    Box(const T& val);              // Конструктор із значенням
    Box(const Box& other);          // Конструктор копіювання
    Box(Box&& other) noexcept;      // Конструктор переміщення
    // Оператор копіювання
    Box& operator=(const Box& other);
    // Оператор переміщення
    Box& operator=(Box&& other) noexcept;
    // Інші методи
    void set(const T& val);
    T get() const;
    void display() const;
};

// Реалізація методів
template <typename T>
Box<T>::Box() : value() {}
template <typename T>
Box<T>::Box(const T& val) : value(val) {}

template <typename T>
Box<T>::Box(const Box& other) : value(other.value) {
  std::cout << "[Копіювання]" << std::endl;
}

template <typename T>
Box<T>& Box<T>::operator=(const Box& other) {
  if (this != &other) {
    value = other.value;
```

```
    std::cout << "[Оператор копіювання]" << std::endl;
  }
  return *this;
}

template <typename T>
Box<T>::Box(Box&& other) noexcept : value(std::move(other.value)) {
  std::cout << "[Переміщення]" << std::endl;
}

template <typename T>
Box<T>& Box<T>::operator=(Box&& other) noexcept {
  if (this != &other) {
    value = std::move(other.value);
    std::cout << "[Оператор переміщення]" << std::endl;
  }
  return *this;
}

template <typename T>
void Box<T>::set(const T& val) {
  value = val;
}

template <typename T>
T Box<T>::get() const {
  return value;
}

template <typename T>
void Box<T>::display() const {
  std::cout << "Box contains: " << value << std::endl;
}
```

Приклад оформлення тексту головної програми:

```
#include "Box.hpp"
#include <string>

int main() {
  Box<std::string> a("Привіт");
  a.display();
  // Копіювання
```

```
    Box<std::string> b = a;                 // Конструктор копіювання
    // Копіювання через оператор =
    Box<std::string> c;
    c = a;                                  // Оператор копіювання
    // Переміщення
    Box<std::string> d = std::move(a);      // Конструктор переміщення
    // Переміщення через оператор =
    Box<std::string> e;
    e = std::move(b);                       // Оператор переміщення
    return 0;
}
```