|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Кафедра математического и программного обеспечения ЭВМ |

# РАЗРАБОТКА АБСТРАКТНЫХ ТИПОВ ДАННЫХ

|  |  |
| --- | --- |
| **Дисциплина:** | ООП |
| **Темы:** | Инкапсуляция; классы; поля и компонентные функции; конструкторы и деструкторы класса |

**Среда разработки:** Microsoft Visual Studio

**Язык программирования:** C++

**Тип проекта:** Консольное приложение

**Задание на лабораторную работу №2**

**ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

1. Каждый класс должен быть оформлен в отдельных файлах: заголовочный (.h) и файл с кодом (.cpp).
2. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
3. Придерживайтесь принципа DRY (Don’t repeat yourself).
4. Обязательно наличие комментариев.

**ЦЕЛЬ РАБОТЫ**

Изучить способы определения классов, правила доступа к элементам; приобрести практические навыки работы с объектами класса; изучить принципы и механизмы создания абстрактных типов данных.

**ЗАДАНИЯ**

1. *часов*
2. Разработайте АТД – класс, обеспечивающий хранение объектов класса из ЛР1, согласно варианту задания (см. раздел прил. 1).
3. Интерфейс класса может содержать конструкторы и должен содержать деструктор.
4. Интерфейс класса должен содержать функции:
   1. Добавления;
   2. Удаления;
   3. Вывода содержимого;
   4. Обработки (сортировка по полю класса, поиск по условию).

**ПРИМЕЧАНИЯ**

* У класса-контейнера не должно быть прямого доступа к полям класса предметной области (ЛР1).

Предметной областью является класс **T**, и его поля, к которым мы хотим получить доступ в классе-контейнере **Tree**. Класс-контейнер **Tree** использует узлы (**Node**), где данные хранятся в указателях на объекты типа **T**. Для доступа к полям предметной области из класса-контейнера нужно использовать методы и функции, предоставленные классом **T**.

**Tree<Cable>** является классом-контейнером, а **Cable** - предметной областью. Когда вставляем объект типа **Cable** в дерево, можем получить доступ к его полям через указатели и методы доступа, предоставленные самим классом **Cable**.

* Amount общая для всех экземпляров и увеличивается с каждым созданным объектом.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Каковы особенности синтаксиса и семантики конструкторов и деструкторов?

**Семантика конструктора**:

* Конструктор вызывается при создании объекта класса.
* Используется для инициализации переменных, резервирования ресурсов и выполнения других необходимых операций.
* Может быть неявно вызван (при объявлении объекта).

**Семантика деструктора**:

* Деструкторы вызываются автоматически при уничтожении объекта.
* Они обычно используются для освобождения ресурсов (например, памяти), занимаемых объектом, и выполнения других завершающих операций.

1. В чем заключаются особенности применения конструкторов по умолчанию?

**Конструктор по умолчанию** - это конструктор, который не принимает аргументов. Если в классе не определён явным образом конструктор, то компилятор может предоставить конструктор по умолчанию.

1. **Автоматическое создание:**

Если в классе не определён ни один конструктор, компилятор автоматически создает конструктор по умолчанию.

1. **Вызов при создании объекта:**

Конструктор по умолчанию вызывается автоматически при создании объекта, если явно не указан другой конструктор.

1. **Инициализация переменных-членов:**

Конструктор по умолчанию инициализирует переменные-члены класса значениями по умолчанию для их типов данных.

1. **Неявное присутствие:**

Если в классе определен хотя бы один конструктор (в том числе пользовательский), конструктор по умолчанию не создается компилятором.

1. В каких случаях вызывается копирующий конструктор?

**Копирующий конструктор** вызывается в следующих случаях:

1. **Прямое присваивание объекта другому объекту**:

Когда объект присваивается другому объекту того же типа.

1. **Передача объекта в функцию по значению**:

Когда объект передается в функцию в качестве параметра по значению (не по ссылке).

1. **Возврат объекта из функции по значению:**

Когда объект возвращается из функции в качестве результата по значению.

1. **Инициализация объекта с использованием другого объекта**:

Когда объект инициализируется с использованием другого объекта того же типа.

1. **Создание копии объекта при передаче через значение в конструктор**:

Когда объект передается в конструктор другого объекта по значению.

1. При каких условиях конструктор может использоваться для преобразования типа?

Конструкторы могут использоваться для преобразования типа в определенных ситуациях, а именно в тех случаях, когда у нас есть одноаргументный конструктор, который может принять значение одного типа и создать объект другого типа. Преобразование типа с использованием конструкторов может быть произведено неявно или явно, в зависимости от того, объявлен ли конструктор как **explicit**.

class Feet {

public:

int length;

Feet(int len) : length(len) {}

};

5. В каких ситуациях необходимы инициализаторы конструктора, синтаксис и семантика их использования?

1. **Константные и ссылочные переменные**:

Когда в классе есть константные переменные или ссылочные переменные, они должны быть инициализированы в списке инициализаторов конструктора, так как они не могут быть присвоены значениями после создания объекта.

1. **Поля с типом данных, не имеющим конструктора по умолчанию:**

Если в классе есть переменные-члены, которые не имеют конструктора по умолчанию и должны быть инициализированы конкретными значениями при создании объекта.

**Приложение 1**

**Варианты:**

1. Статический вектор\*

2. Динамический вектор\*

3. Статическая матрица\*\*

4. Динамическая матрица\*\*

5. Линейный односвязный список\*

6. Линейный двусвязный список\*

7. Стек\*

8. Очередь\*

9. Закольцованный список\*

10. Двоичное дерево\*

\* - обязательно с использованием структуры и указателей.

\*\* - разрешается использовать двумерный динамический массив.