Экзаменационная программа по курсу «Программирование на C++. Часть 2»

1. Основы создания классов. Инкапсуляция

- Отличие C++ от C: включение функций в структуры (классы) как добавление поведения к характеристикам.
- Понятие класса, объекта (экземпляра класса), полей и методов.
- Инкапсуляция: объединение данных и методов, управление доступом.
- Спецификаторы доступа: private, protected, public.
- Различие между struct и class (доступ по умолчанию).
- Рекомендации по проектированию класса: поля – закрыты, интерфейс – открыт, служебные функции – закрыты.
- Указатель this: неявный параметр функций-членов, разрешение коллизий имён.

2. Конструкторы и деструкторы

- **Конструкторы:** назначение, имя, отсутствие возвращаемого значения.
- Конструктор без параметров (конструктор по умолчанию).
- Конструктор с параметрами.
- Конструктор со списком инициализаторов (: field(value), ...). Преимущества (эффективность, инициализация констант и ссылок).
- Конструктор с параметрами по умолчанию.
- Неявный конструктор по умолчанию: условия генерации, опасность неочевидных ошибок (исчезновение после объявления любого другого конструктора).
- Конструктор копирования: понятие, когда вызывается (явное создание копии, передача по значению, возврат по значению).
- Проблема автоматического конструктора копии (поверхностное копирование vs глубокое копирование) для классов с динамической памятью.
- **Деструктор:** имя (~), отсутствие аргументов, назначение (освобождение ресурсов).
- Неявный деструктор.
- Порядок вызова деструкторов (при уничтожении объекта).

3. Перегрузка операций

- Понятие перегрузки операций как способа придания операциям нового смысла для пользовательских типов.
- Синтаксис: operator@.
- Два способа перегрузки: как функция-член и как внешняя функция.
- Рекомендации по выбору способа (изменяет левый аргумент -> член; симметрия операндов или cout << obj -> внешняя).
- Перегрузка бинарных операций (+, +=, ==, !=).

- Перегрузка унарных операций (префиксный и постфиксный ++, --): фиктивный параметр int для постфиксной формы.
- Перегрузка операций ввода/вывода (<<, >>) как внешних дружественных функций, возврат ссылки на поток.
- Перегрузка оператора индексации []: возврат по ссылке, константная и неконстантная версии.
- Перегрузка оператора приведения типа (operator double(), operator char*()).
- Ограничения перегрузки операций (нельзя перегрузить ::, ., sizeof, ?: и т.д.).

4. Управление ресурсами. Правило трёх и пяти

- **Правило трёх:** Если классу требуется явный деструктор, конструктор копирования или оператор присваивания копированием, то, скорее всего, нужны все три.
- Оператор копирующего присваивания (operator=): проверка на самоприсваивание (this != &other), корректное освобождение старого ресурса и выделение нового.
- Исключения в операторе присваивания: базовая гарантия, строгая гарантия (использование локальной копии и swap).
- Идиома **copy-and-swap**.
- **Семантика перемещения (C++11):** rvalue-ссылки (&&), конструктор перемещения, оператор присваивания перемещением.
- **Правило пяти:** деструктор, копирующий конструктор, копирующее присваивание, перемещающий конструктор, перемещающее присваивание.

5. Наследование и полиморфизм

- **Наследование:** отношение "is-a". Базовый и производный классы.
- Виды наследования (public, protected, private). Таблица доступности членов.
- Порядок вызова конструкторов и деструкторов при наследовании.
- Переопределение методов. Отличие переопределения от перегрузки.
- Совместимость типов: указатель/ссылка на базовый класс могут указывать на объект производного.
- **Полиморфизм:** статическое (раннее) и динамическое (позднее) связывание.
- **Виртуальные функции** (virtual): механизм позднего связывания.
- **Чисто виртуальные функции** (= 0). Абстрактные классы (нельзя создать объект).
- Таблица виртуальных методов (VMT), указатель vptr.
- Виртуальный деструктор (правило: если есть хоть одна виртуальная функция, деструктор должен быть виртуальным).
- **Преобразование типов в иерархии:**
    - upcast (от наследника к базовому) – неявное.
    - downcast (от базового к наследнику) – явное, static_cast, dynamic_cast.

- dynamic_cast**:** безопасное приведение для полиморфных типов, возвращает nullptr для указателей или генерирует std::bad_cast для ссылок.
- **RTTI (Runtime Type Information):** typeid, std::type_info.
- Множественное наследование, проблема ромба (diamond problem), виртуальное наследование.
- Спецификаторы override и final.
- Закрытое наследование (private) как отношение "as-a" (реализовано посредством).

6. Дружественность (friend)

- Дружественные функции (доступ к закрытым членам класса).
- Дружественные классы.
- Сравнение дружественных функций с функциями доступа (геттерами/сеттерами).

7. Статические члены класса

- Статические поля: одна копия для всех объектов, инициализация вне класса.
- Статические методы: не имеют указателя this, могут обращаться только к статическим членам.
- Подсчёт количества объектов с помощью статического поля.

8. Шаблоны (Templates)

- Шаблоны функций и шаблоны классов.
- Параметры шаблонов: типы (typename/class), нетиповые параметры (константы), параметры-шаблоны.
- Специализация шаблонов (полная и частичная).
- Явная специализация.
- Особенности компиляции шаблонов (код в заголовочных файлах).
- Инстанцирование шаблона. Ключевое слово typename.
- Конструкторы преобразования и ключевое слово explicit.

9. Обработка исключений (Exception Handling)

- Конструкция try, catch, throw.
- Порядок обработчиков catch (от частного к общему, catch(...)).
- Исключения в конструкторах и деструкторах.
- Гарантии безопасности исключений (базовая, строгая, отсутствие).

10. Умные указатели (Smart Pointers) – C++11 и новее

- Проблемы с сырыми указателями: утечки памяти, висячие ссылки.
- std::unique_ptr: эксклюзивное владение, запрет копирования, перемещение через std::move.

- std::shared_ptr: подсчёт ссылок, совместное владение.
- std::weak_ptr: предотвращение циклических ссылок (shared_ptr), метод lock().
- Фабричные функции: std::make_shared, std::make_unique.
- Проблема std::auto_ptr (deprecated).

11. Отношения между классами

- Наследование ("is-a").
- Включение/композиция ("has-a").
- Закрытое наследование ("as-a" или "реализовано через").

12. STL (Standard Template Library)

- Основные компоненты STL: контейнеры, итераторы, алгоритмы, адаптеры, функторы, аллокаторы.
- **Контейнеры:**
    - Последовательные: vector, list, deque, array.
    - Ассоциативные: set, multiset, map, multimap.
    - Неупорядоченные: unordered_set, unordered_map.
    - Адаптеры контейнеров: stack, queue, priority_queue.
- **Итераторы:** понятие, категории (ввода, вывода, однонаправленные, двунаправленные, произвольного доступа).
- Функции begin(), end(), rbegin(), rend().
- Итераторы вставки (back_inserter, front_inserter, inserter).
- Потоковые итераторы (istream_iterator, ostream_iterator).
- **Алгоритмы STL (основные):**
    - Немодифицирующие: find, find_if, count, count_if, for_each, search, equal.
    - Модифицирующие: copy, copy_if, transform, replace, remove, unique, reverse.
    - Сортировка и операции с множествами: sort, set_intersection, set_union.
- **Функторы (функциональные объекты)** и лямбда-выражения (C++11): синтаксис [](){}, списки захвата ([=], [&], [a], [&b]), mutable.
- Класс std::function.

13. Продвинутые темы C++

- **constexpr:** вычисление на этапе компиляции, отличие от const.
- **lvalue и rvalue:** понятие, ссылки на rvalue (&&).
- **Ковариантные возвращаемые типы** виртуальных функций.
- Управление генерацией специальных функций: = default, = delete.

- **Синглтон:** понятие, запрет копирующего конструктора и оператора присваивания.

14. Практические задачи (примеры из лекций)

- Реализация класса Date.
- Реализация класса Circle.
- Реализация класса Array (динамический массив).
- Реализация класса String.
- Реализация классов Person – Student (наследование).
- Реализация классов геометрических фигур с полиморфизмом (shape, circle, rectangle).
- Реализация класса-счётчика объектов (Counter).
- Реализация класса дроби (frac).
- Реализация бинарного дерева поиска (рекурсивные структуры).
- Реализация двусвязного списка и итератора к нему.
- Реализация шаблонного класса Stack.
- Реализация класса Matrix на основе vector<vector>.
- Использование map для частотного словаря.
- Использование set и set_intersection для поиска студентов-задолжников.