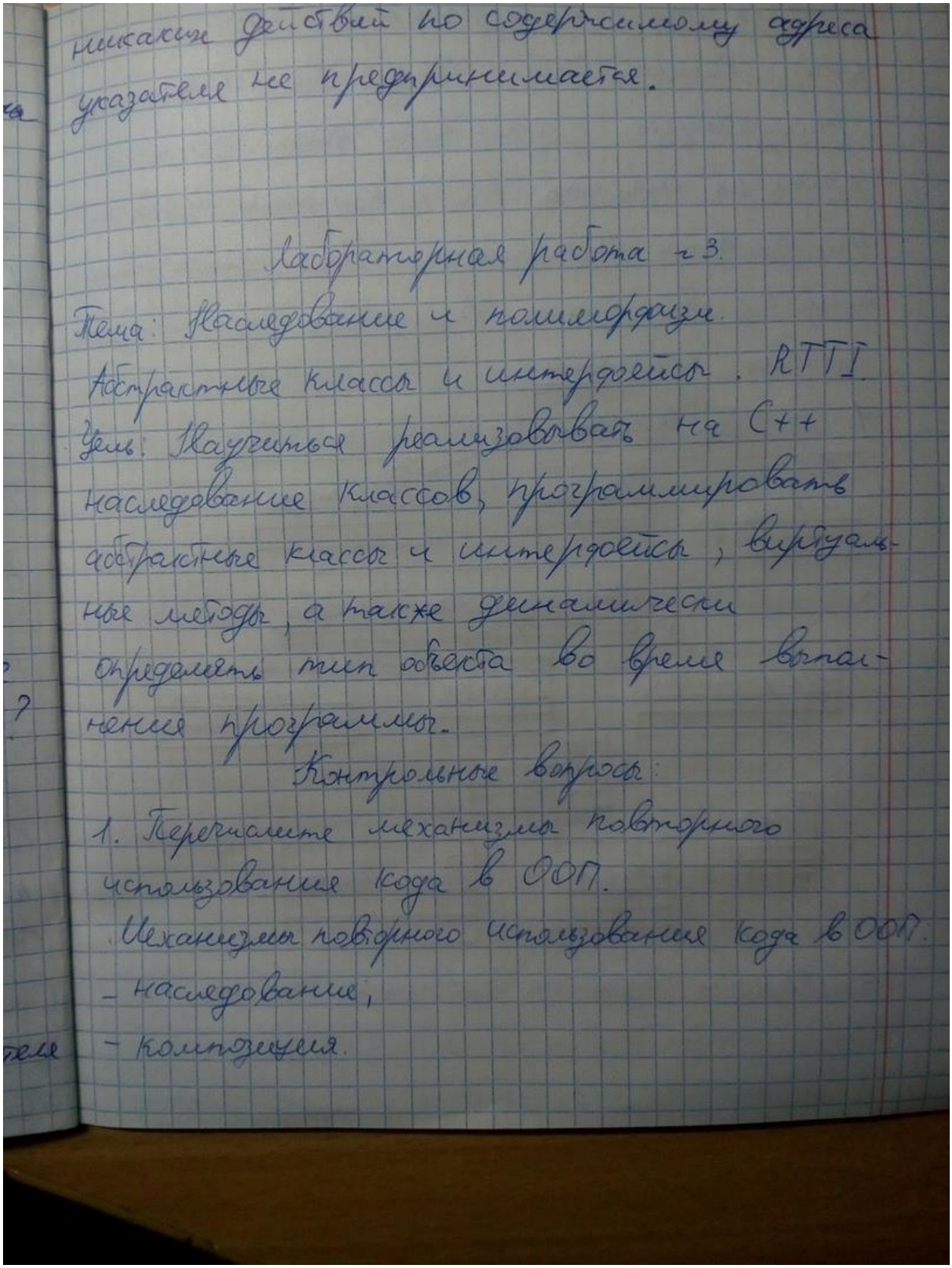
Хайтул П.А. 1(уск) ИВТ-5
Ответы на контрольные вопросы:



никаких действий по содержимому адреса указателя не предпринимается.

Лабораторная работа №3.

Тема: Наследование и полиморфизм. Абстрактные классы и интерфейсы. RTTI.

Цель: Научиться реализовывать на C++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.

Контрольные вопросы:

1. Перечислите механизмы повторного использования кода в ООП.

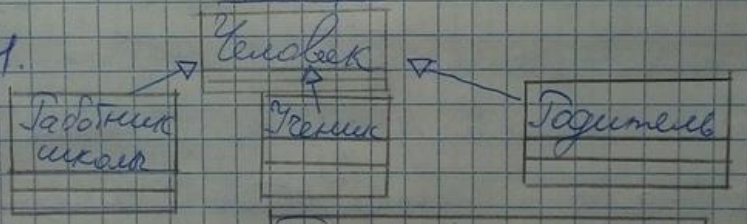
Механизмы повторного использования кода в ООП:

- наследование;
- композиция.

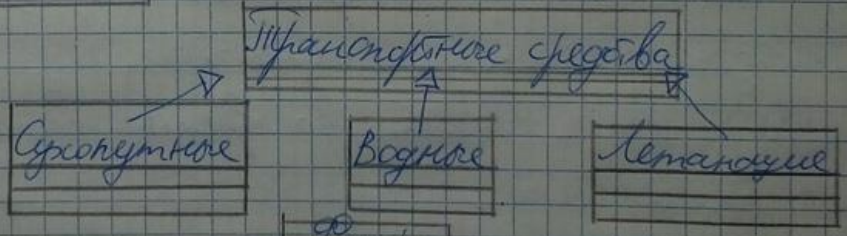
2. Что такое наследование? Приведите три примера схем наследования классов.

Наследование представляет собой один из ключевых аспектов ООП, который позволяет наследовать функциональность одного класса или базового класса в другом - производном классе. Примеры:

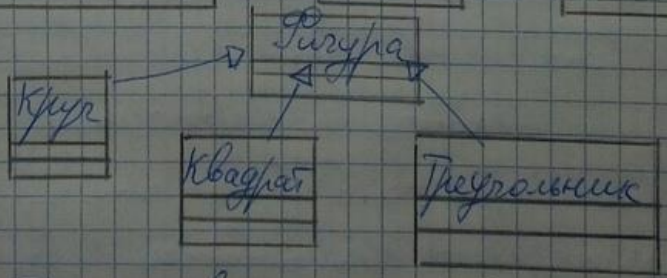
1.



2.



3.



3. Типы и виды наследования в C++.

Для спецификации доступа в C++ есть три уровня: private, protected и public. Аналогично типы наследования:

- private-наследование - данные protected и public базового класса доступны из методов производного класса, но недоступны извне, то есть они становятся private;
- protected-наследование - данные protected и public базового класса становятся protected;
- public-наследование - публичное наследование наиболее употребительно. При этом protected и public данные из базового класса остаются, соответственно protected и public в производном классе.

	Исходный модификатор доступа		
	public	private	protected
public-наследование	public	private	protected
private-наследование	private	private	private
protected-наследование	protected	private	protected

Виды наследования:

- одиночное (простое) наследование – производный класс наследует элементы одного базового класса;
- множественное наследование – производный класс наследует элементы более, чем от одного базового класса.

4. В чем заключается разница между ранним и поздним связыванием?

Раннее связывание означает, что компилятор может напрямую связать имя идентификатора (например, имя функции или переменной) с машинным адресом

Позднее связывание - в некоторых программах невозможно знать наперед, какая функция будет вызываться первой. В языке C++ для выполнения позднего связывания используются указатели на функции.

Указатель на функцию - тип указателя, который указывает на функцию вместо переменной.

5 Что такое полиморфизм? Виртуальные методы

Полиморфизм - свойство, которое позволяет использовать одно и тоже имя функции для решения двух и более схожих, но технически разных задач. Полиморфизм - возможность замещения методов в объекта родителя методами объекта-потомка, имеющих то же имя.

Чтобы использовать полиморфизм, необходимо чтоб:

1) все классы-потомки являлись наследниками одного и того же базового класса.

2) функция, реализующая метод, должна быть объявлена виртуальной в базовом классе.

Виртуальная функция - специальный вид функций-членов класса. Виртуальная функция отличается от обычной функции тем, что для обычной функции связывание вызова функции с её определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время

выполнения программы.
Для объявления виртуальной функции
используется ключевое слово virtual.
Функция-член класса может быть
объявлена как виртуальная, если
- класс, содержащий виртуальную функцию,
базовый в иерархии порождения;
- реализация функции зависит от класса
и будет различной в каждом порожденном
классе.
Виртуальная функция - это функция, которая
определяется в базовом классе, а любой
порожденный класс может ее переопределить.
Виртуальная функция вызывается только через
указатель или ссылку на базовый класс.
6. Что такое абстрактный класс? Чисто
виртуальные методы.
Абстрактные классы - это классы, которые
содержат или наследуют без переопределения
хотя бы одну чисто виртуальную функцию.

Абстрактный класс определяет интерфейс для наследования производными классами. Чисто виртуальная функция - функция, которая не имеет определения. Чтобы определить виртуальную функцию как чисто, ее объявление завершается значением "=0". Например:

```
class Figure
{
public:
    virtual double getSquare() = 0;
    virtual double getPerimeter() = 0;
    virtual void ShowFigureType() = 0;
};
```

7. Что из себя представляет класс-интерфейс? Для чего он используется?

Интерфейс - это класс, который не имеет переменных-членов и все методы которого являются чисто виртуальными функциями. Использование интерфейсов - один из вариантов

обеспечения полиморфизма в объектных языках и средах. Все классы, реализующие один и тот же интерфейс, с точки зрения определяемого ими поведения, ведут себя вполне одинаково. Это позволяет писать обобщенные алгоритмы обработки данных, использующие в качестве типов параметра интерфейсов, и применять их к объектам различных типов, всякий раз получая предсказуемый результат. Интерфейсы позволяют наладить множественное наследование классов.

8. Порядок создания и удаления подклассов. Зачем нужен виртуальный деструктор?

Деструкторы вызываются в том же самом порядке, в каком классы следуют один за другим в иерархии классов. В противоположность этому деструктор производного класса вызывается перед деструктором базового класса, т.е. подклассы ~~следовательно~~, будут удалены в обратном порядке

Если в классе присутствует хотя бы одна виртуальная функция, деструктор также следует сделать виртуальным.
Так обеспечивается корректное разрушение объекта производного класса через указатель на соответствующий базовый класс.

9. Что означает RTTI? Какие есть возможности RTTI в C++?

Динамическая идентификация типа данных (RTTI) - механизм, который позволяет определить тип данных переменной или объекта во время выполнения программы. Для этого используется функция typeid. Для использования этой функции необходимо включить заголовочный файл typeinfo.h
Запись typeid: typeid(объект),
здесь объект является объектом, чей тип требуется определить. Функция typeid возвращает ссылку на объект типа typeinfo, который описывает тип объекта.
Преобразование типов в C++:
- static_cast<> - для приведений, которые проверяются только во время компиляции, возвращает ошибку, если компилятор обнаруживает что-то, что пытается выполнить приведение типов, которое полностью несовместимо.

- dynamic_cast <> - для безопасного выполнения приведения указателя на Base к типу, которая проверяется средой. Объект dynamic_cast> является более безопасным static_cast> чем для типов, но проверка среды выполнения требует некоторых дополнительных издержек;
- const_cast < > - для приведения параметра const - здания характеристики переменной или const для преобразования неперемещаемой в значение const;
- reinterpret_cast < > - для приведения между несвязными типами, такими как тип указателя и int.