1. **Своими словами развёрнуто ответить на вопросы:**
   1. **Что послужило толчком к созданию парадигмы ООП?**

-программы становились слишком сложными, а их надо было как-то модифицировать.

-создание больших программ требовало обсуждений внутри коллектива имен функций, переменных и тому подобного, это было очень неудобно.

* 1. **Экскурс в историю: в каком языке были разработаны основные принципы ООП? Дайте краткий обзор этого языка.**

-Основа ООП была заложена в начале 1960-х годов. Прорыв в использовании экземпляров и объектов был достигнут в MIT с PDP-1, и первым языком программирования для работы с объектами стал Simula 67.

- Это язык для имитационного моделирования реальности (по сути общего назначения). Является «дедушкой» Паскаля, так что к использованию сейчас непригоден.

* 1. **Подробно опишите показатели качества декомпозиции программы**

**-** сцепление внутри компонента

Степень взаимосвязи отдельных его частей (если внутри

компонента несколько подзадач, которых легко можно разделить, то у компонента слабое сцепление)

- связанность между компонентами

Интерфейс между компонентом-клиентом и компонентом-сервером. Общее число входов и выходов сервера, это мера связанности. Чем меньше связанность между двумя компонентами, тем проще понять и отслеживать в будущем их взаимодействие.

* 1. **Назовите несколько отличий языков C и С++.**

**-** В C++ есть ссылки, то есть можно передавать значения через вот это &, я не через указатель \*.

**-**В С нет операций декрементации(--) и инкрементации(++)

-В Си++ придумали тип bool, который может иметь два значения — «false» (ложь) и «true» (истина), а в С были зачения нуля и не нуля.

-В С++ появились new и delete для работы с динамической памятью.

-В C++ появились классы и объекты. (Это как структуры и переменные в С, но в C++ есть еще модификаторы доступа и полями могут быть не только данные, но и функции)

* 1. **Подробно опишите идентификаторы (где начало и конец области действия, какова видимость, каково время жизни), описанные в блоке, файле, функции, прототипе функции, классе и поименованной области.**

**В блоке:**

-Начало при объявлении в блоке.

-Конец, когда заканчивается блок.

-Видно только в блоке.

-Время жизни, пока программа идет через блок.

**В файле:**

- Начало при объявлении (открытии файла).

- Конец при закрытии файла.

- Виден везде (кроме функций, в которые не передается).

- Живет до (закрытия файла).

**В функции:**

- Начало с момента объявления.

- До конца функции.

-Видна только после объявления в программе.

- Живет каждый раз при вызове (то время, что исполняется)

**В прототипе функции:**

Этот компонент используется для того, чтобы компилятор знал о существовании функции до её определения, то есть по сути обладает теми же свойствами, что и функция.

**В классе:**

- Начало при создании объекта, к которому относится.

- Конец при уничтожении объекта.

- Если создан в основном теле программы, то виден везде.

- Живет, пока вызванный объект существует в программе.

**В поименованной области:**

- Начало с момента объявления.

- До конца программы.

- Видим с момента объявления везде.

- Живет до конца программы.

* 1. **Чем отличаются заголовочные файлы stdio.h и cstdio?**

Заголовочный файл cstdio объявляет свои элементы в пространстве имен std, а stdio.h объявляет те же элементы в глобальном пространстве имен. Чтобы обратиться к глобальному пространству имен, нужно использовать оператор «::» (доступа в области видимости)

* 1. **В чём структурный подход к программированию уступает объектно-ориентированному подходу?**

-повышение степени абстракции программы (внутри программы стало проще взаимодействие объектов друг с другом, программа стала похожа на подобие человеческой речи, где информация об объекте вкладывается в его название.  
-повышение надёжности программ (то, что и написано)  
-облегчение модификации программы (то, что и написано)

-в структурном программировании для чего-то, похожего на наследование пришлось бы как бы копировать все свойства одного объекта в другой, а в ООП при наследовании свойства не описываются повторно.

1. **Составьте программу (из одного модуля), в которой один и тот же идентификатор встречается 9 раз (метод вложенных блоков не принимается).**

#include <iostream>

using namespace std;

void func() {

int identificator=2; // 2

}

namespace identificator // 5 на эти названия компилятор вообще никак не реагирует

{

int identificator(int a, int b) // 3

{

return a + b;

}

}

namespace hihihi

{

int shtoto(int a, int b)

{

float identificator=4; // 4

return a + b-identificator;

}

}

class Monster {

int identificator, ammo; // 6

public:

Monster(int he=100, int am=10) {

identificator = he; ammo = am;}

int GetHealth() {return identificator;}

int GetAmmo() {return ammo;}

void Draw(int x, int y);

};

namespace hihihi

{

class identificator { // 8

int hehehe, ammo; // 6

public:

identificator(int he=100, int am=10) {

hehehe = he; ammo = am;}

int GetHealth() {return hehehe;}

int GetAmmo() {return ammo;}

void Draw(int x, int y);

};

}

int funct() {

FILE \*identificator; // 9

identificator=fopen("s.txt", "wb");

fclose(identificator);

}

int main()

{

int identificator=1; // 1

int a=0;

for (int identificator=7;identificator<9;identificator++) { // 7

a++;

cout<<a<<identificator<<endl;

}

cout<<a<<identificator<<endl;

funct();

}

Результат выполнения работы данной программы

