**Отчет**

Входе работы рассмотрела и применила на практике следующие концепции:

* как происходит создание и разрушение объектов при их инициализации и завершении работы.
* познакомилась с механизмами копирования объектов при передаче их в функции по значению, понимая, что создается копия объекта с теми же значениями полей.
* Изучила последовательность инициализации и уничтожения составных объектов, а также их базовых классов или полей в композиции.
* Освоила методы размещения и использования статических полей класса, которые существуют в единственном экземпляре на уровне класса.
* Познакомилась с правилами работы с динамическим выделением памяти и освобождением её после использования, чтобы избежать утечек памяти и ошибок.
* Изучила возможности шаблонов для создания универсальных классов и функций, способных работать с различными типами данных без явного указания.

a) В C++ существуют два случая неявного копирования объектов:

Когда объект передается в функцию по значению.

Когда объект возвращается из функции по значению.

b) В C++ составные объекты создаются в следующем порядке:

Сначала вызывается конструктор базового класса.

Затем вызываются конструкторы членов класса в порядке их объявления в классе.

После этого вызывается конструктор самого класса.

c) Интерфейс IDisposable в C# предназначен для реализации управления ресурсами. Он содержит метод Dispose(), который позволяет освободить ресурсы, занимаемые объектом, и выполнять другие очисточные операции.

d) В C# метод Dispose не будет вызван по завершению блока using, если на объект, указанный в using, имеется еще одна ссылка. В этом случае ресурсы будут освобождены только при вызове метода Dispose явно или при уничтожении объекта сборщиком мусора.

Пример кода на C#:

using System;

class Test : IDisposable

{

public void Dispose()

{

Console.WriteLine("Disposed");

}

}

class Program

{

static void Main(string[] args)

{

Test test = new Test();

using (test)

{

Console.WriteLine("Inside using block");

} // Здесь Dispose не будет вызван, так как на объект test еще есть ссылка

Console.WriteLine("Outside using block");

}

}

e) Для вызова конструктора предка с параметром в C++ можно использовать список инициализации в конструкторе потомка:

class Parent {

public:

Parent(int param) {

// конструктор предка с параметром

}

};

class Child : public Parent {

public:

Child(int param) : Parent(param) {

// конструктор потомка

}

};

f) В C# вызов конструктора предка с параметром осуществляется с помощью ключевого слова base:

class Parent {

public Parent(int param) {

// конструктор предка с параметром

}

}

class Child : Parent {

public Child(int param) : base(param) {

// конструктор потомка

}

}

g) В C++ вызов конструктора члена класса с параметром осуществляется также с помощью списка инициализации в конструкторе агрегата:

class Member {

public:

Member(int param) {

// конструктор члена класса с параметром

}

};

class Aggregate {

private:

Member member;

public:

Aggregate(int param) : member(param) {

// конструктор агрегата

}

};

h) В C++ ссылка - это альтернативное имя для объекта или переменной, которое используется во время объявления и не может быть изменено после инициализации. В .NET ссылка - это адрес в памяти, который указывает на местонахождение объекта.

i) Шаблон класса в C++ объявляется с использованием ключевого слова template, за которым следует список параметров шаблона в угловых скобках <>, а затем само объявление класса. Пример использования шаблона класса:

template <typename T>

class MyTemplate {

public:

T value;

MyTemplate(T val) : value(val) {}

};

int main() {

MyTemplate<int> myInt(5);

MyTemplate<double> myDouble(3.14);

return 0;

}

В C# шаблон класса называется обобщенным классом и объявляется с использованием угловых скобок <> после имени класса. Пример использования обобщенного класса:

class MyGenericClass<T> {

public T value;

public MyGenericClass(T val) {

value = val;

}

}

class Program {

static void Main(string[] args) {

MyGenericClass<int> myInt = new MyGenericClass<int>(5);

MyGenericClass<double> myDouble = new MyGenericClass<double>(3.14);

}

}