1. Основные алгоритмические конструкции в языке программирования

Как вам уже известно из курса информатики, любой алгоритм может быть записан с использованием трех базовых алгоритмических конструкций: следование, цикл и ветвление

1. Структура программы на языке C#

Структура программы на языке C# обычно состоит из следующих элементов:

1. **Пространства имен (Namespaces)**: Используются для организации кода и предотвращения конфликтов имен. Обычно объявляются в начале файла.
2. **Директива using**: Используется для импортирования пространств имен, чтобы обеспечить доступ к классам и методам, определенным в этих пространствах имен.
3. **Основной класс (Main class)**: Обычно содержит метод Main(), который является точкой входа в программу.
4. **Метод Main()**: Этот метод является начальной точкой выполнения программы. Здесь выполняются основные инструкции программы.
5. **Другие классы и методы**: Кроме основного класса, программа может содержать дополнительные классы и методы, которые выполняют различные функции в программе.
6. **Переменные и константы**: Объявляются для хранения данных и значений, которые используются в программе.
7. **Управляющие структуры**: Как и в других языках программирования, в C# используются условные выражения (if-else), циклы (for, while, do-while) и т.д., для управления потоком выполнения программы.
8. **Модификаторы доступа**: Определяют уровень доступа к классам, методам и переменным (например, public, private, protected).
9. **Комментарии**: Используются для документирования кода и делают его понятным для других разработчиков.
10. Понятие переменной и константы

Переменная - это именованное место в памяти, которое используется для хранения данных.

Константа — это постоянное значение, которое известно во время компиляции и не изменяются во время выполнения программы. Константы объявляются модификатором const и инициализируются сразу после объявления с помощью оператора следующей формы: const тип имя\_константы = значени

1. Понятие типа данных

Вот некоторые общие типы данных:

1. **Целочисленные типы (integers)**: Хранят целые числа без дробной части. Например, int, long, short, byte.
2. **Вещественные типы (floating-point numbers)**: Используются для представления чисел с плавающей точкой, то есть чисел с дробной частью. Например, float, double, decimal.
3. **Символьные типы (characters)**: Хранят одиночные символы. Например, char.
4. **Строковые типы (strings)**: Хранят последовательности символов. Например, string.
5. **Логический тип (boolean)**: Хранит значение истинности (true или false). Например, bool.
6. **Составные типы (composite types)**: Включают в себя более сложные структуры данных, такие как массивы, списки, словари, объекты и т. д. Они могут содержать комбинации других типов данных.
7. **Пользовательские типы (user-defined types)**: Типы данных, определенные пользователем. Это может быть класс, структура или перечисление, которые определяются в программе.
8. Консольный ввод данных
9. Консольный вывод данных
10. Оператор присваивания
11. Полный условный оператор

if (условие)

{

// Код, который будет выполнен, если условие истинно

}

else

{

// Код, который будет выполнен, если условие ложно

}

1. Неполный условный оператор

if (условие)

{

// Код, который будет выполнен, если условие истинно

}

1. Оператор выбора

switch (выражение)

{

case значение1:

// Код, который будет выполнен, если выражение равно значению1

break;

case значение2:

// Код, который будет выполнен, если выражение равно значению2

break;

// Дополнительные случаи

default:

// Код, который будет выполнен, если ни одно из значений не соответствует выражению

break;

}

1. Цикл с предварительной проверкой условия

while (условие)

{

// Код, который будет выполняться, пока условие истинно

}

1. Цикл с последующей проверкой условия

do

{

// Код, который будет выполняться хотя бы один раз

// затем условие будет проверено

} while (условие);

1. Цикл с параметром

for (инициализация; условие; итерация)

{

// Код, который будет выполняться в цикле

}

1. Понятие массива и массива объектом

int[] numbers = new int[5];

Person[] people = new Person[3];

1. Понятие объектно-ориентированного программирования

В ООП мы думаем о программировании как о работе с объектами, которые могут делать определенные вещи и иметь свои свойства. Вместо того чтобы рассматривать программу как набор инструкций, мы представляем ее как набор взаимодействующих объектов.

Представьте, что у вас есть объект "Собака". У этого объекта могут быть свойства, такие как "имя", "возраст" и "порода". У этого объекта также могут быть методы, такие как "сказать\_гав" и "бегать". Таким образом, в ООП мы сосредотачиваемся на том, что объекты могут делать (методы) и что у них есть (свойства).

1. Основные принципы ООП. Знать определения каждого принципа

Основные принципы объектно-ориентированного программирования включают:

1. **Инкапсуляция**: Идея заключается в объединении данных и методов, которые манипулируют этими данными, в единый объект. Инкапсуляция позволяет скрыть детали реализации и предоставить интерфейс для работы с объектом.
2. **Наследование**: Это механизм, который позволяет новому классу (подклассу) наследовать свойства и методы существующего класса (родительского класса). Наследование позволяет повторно использовать код и создавать иерархии классов.
3. **Полиморфизм**: Полиморфизм означает, что один и тот же метод или оператор может иметь разные формы. Это может быть реализовано через перегрузку методов (overloading), виртуальные методы (virtual methods), абстрактные классы и интерфейсы.
4. Определение класса

Класс (lat. «группа») – это множество объектов, имеющих общую структуру и общее поведение

1. Определение объекта класса

Объект — это некоторая сущность, обладающая определенным состоянием и поведением

1. Что значит описать объект?

Что означает описать объект? – это определить его свойства и методы

1. Что же значит создать объект?

Что означает создать объект? – это обратиться к конкретному экземпляру описанного объекта

1. Этапы создания объекта

1) Создали класс 2) Создали экземпляр класса (объект) 3) Обращаемся к свойствам и методам экземпляра класса

1. Понятие экземпляра класса

Экземпляр класса, также известный как объект, представляет собой конкретное воплощение класса в программе.

1. Структура описания класса

public class Person

{

// Поля

private string name;

private int age;

// Свойства

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

// Конструктор

public Person(string name, int age)

{

this.name = name;

this.age = age;

}

// Метод

public void SayHello()

{

Console.WriteLine("Привет, меня зовут " + name + " и мне " + age + " лет.");

}

}

1. Понятие поля

Поле в объектно-ориентированном программировании (ООП) представляет собой переменную, объявленную внутри класса

1. Модификаторы доступа элементов класса

Различные модификаторы доступа (public, private, protected, internal)

1. Область видимости (контекст) переменных и констант

Область видимости определяет, где переменные и константы могут быть использованы:

1. Локальная область видимости: Переменные, определенные внутри метода, доступны только внутри этого метода.

2. Глобальная область видимости: Переменные и константы, объявленные в классе, доступны из любого метода этого класса.

3. Параметры метода: Параметры метода доступны только внутри этого метода.

4. Область видимости констант: Константы доступны из любой части кода, где они были объявлены.

1. Понятие метода

Метод в объектно-ориентированном программировании (ООП) - это функция или процедура, объявленная внутри класса, которая определяет поведение объекта этого класса. Методы предоставляют способ для выполнения операций над данными объекта или выполнения каких-либо действий в контексте этого объекта.

1. Описание метода

Описание метода включает в себя указание его сигнатуры (имени, параметров и типа возвращаемого значения), а также описания его назначения и действия, которые он выполняет.

1. Доступ к методу

Доступ к методу зависит от его модификатора доступа.

1. Определение конструктора

Конструктор в объектно-ориентированном программировании (ООП) - это специальный метод, который вызывается при создании объекта класса. Его основная задача состоит в инициализации нового объекта.

1. Особенности конструктора
2. Назначение конструктора

Его основная задача состоит в инициализации нового объекта.

1. Общая форма записи конструктора

public Person() { Name = "Unknown"; Age = 0; }

1. Виды конструкторов и их отличие

Конструкторы в объектно-ориентированном программировании бывают двух видов:

1. \*\*Конструктор без параметров\*\*: Создается по умолчанию, не принимает аргументов и инициализирует объект значениями по умолчанию.

2. \*\*Конструктор с параметрами\*\*: Принимает аргументы и используется для инициализации объекта значениями, переданными при создании экземпляра класса.

Конструкторы позволяют создавать объекты с определенными начальными значениями.

1. Определение интерфейса

Интерфейс в объектно-ориентированном программировании (ООП) - это абстрактный тип данных, который определяет набор методов и свойств, которые класс должен реализовать.

1. Объявление интерфейса

public interface IShape

{

// Метод для вычисления площади фигуры

double CalculateArea();

// Метод для вычисления перимет

double CalculatePerimeter();

// Свойство для получения или установки цвета фигуры

string Color { get; set; }

}

1. Общая форма реализации интерфейса в классе

public class MyClass : IShape

{

// Реализация метода интерфейса

public double CalculateArea()

{

// Код для вычисления площади

}

// Реализация метода интерфейса

public double CalculatePerimeter()

{

// Код для вычисления периметра

}

// Реализация свойства интерфейса

public string Color { get; set; }

}

1. Особенности интерфейсов

Кратко об особенностях интерфейсов:

1. **Абстрактность**: Определяют только сигнатуры методов, не реализацию.
2. **Обязательство**: Классы, реализующие интерфейс, должны предоставить реализацию его методов.
3. **Множественное наследование**: Классы могут реализовать несколько интерфейсов.
4. **Разделение интерфейса и реализации**: Разделяют спецификацию от реализации.
5. **Поддержка стандартов**: Используются для стандартизации поведения классов.
6. **Использование в полиморфизме**: Позволяют использовать одинаковый интерфейс для объектов различных классов.
7. **Разделение кода на модули**: Используются для разделения кода на независимые модули.
8. Отличия между интерфейсами и абстрактными классами

Краткие отличия между интерфейсами и абстрактными классами:

1. **Реализация методов**: Абстрактные классы могут иметь как абстрактные, так и конкретные методы, в то время как интерфейсы содержат только сигнатуры методов, без их реализации.
2. **Наследование**: Класс может наследовать только один абстрактный класс, но может реализовать несколько интерфейсов.
3. **Состояние**: Абстрактный класс может содержать поля (состояние), а интерфейс не может.
4. **Гибкость**: Использование интерфейсов обеспечивает большую гибкость и уменьшает связанность в сравнении с абстрактными классами.
5. **Наследование от других классов**: Абстрактные классы могут быть использованы как базовые классы для других классов, в то время как интерфейсы не имеют такой возможности.
6. Какие элементы языка программирования C# можно объявлять статическими?

Элементы в C#, которые могут быть объявлены статическими: поля, методы, свойства, конструкторы, классы и вложенные классы.

1. Понятие статического класса. Какие особенности использования статических классов в программах на C#?

Статический класс в C# - это класс, который содержит только статические члены (поля, методы, свойства), и не может быть инстанциирован.

1. Понятие свойства в C#

Свойство – это специальный тип членов класса, который предназначен для организации доступа к закрытым полям класса и определяет методы его получения и установки (для контроля чтения и записи полей).

1. Общая форма (синтаксис) объявления свойства в классе

[модификаторы] тип Имя

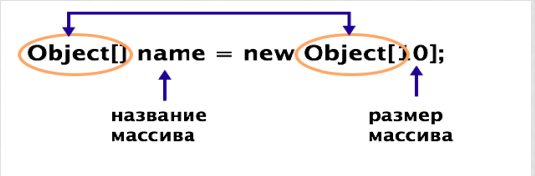
{

{ get return field; } // получение значения свойства

{ set field=value; } // установка значения свойства

}

1. Объявление массива объектов



1. Алгоритм создания массива объектов

1) объявление класса; 2) объявление ссылочной переменной на массив (рис. 1а); 3) выделение памяти для массива ссылок на объекты и сохранение адреса выделенной памяти в ссылочной переменной (рис. 1б); 4) выделение памяти для объектов и запись в соответствующие элементы массива ссылок на эти объекты (рис. 1в).

1. Назначение метода Main()

Метод Main() в C# является точкой входа в приложение.

1. Определение структуры

Структура (struct) в C# — это пользовательский тип данных, который используется наряду с классами и может содержать какие-либо данные и методы.

1. Синтаксис структуры

struct Имя структуры

{

// элементы структуры (поля, методы и т.д.)

}

1. Этапы использования структуры в программе

Использование структуры в программе происходит в 2 этапа:

1. объявление типа структуры;

2. объявление структурной переменной.

1. Отличия между структурами и классами

Отличия между структурами и классами в C#:

1. **Семантика копирования**: Структуры копируются по значению, тогда как классы - по ссылке.
2. **Наследование**: Классы поддерживают наследование, а структуры - нет.
3. **Область видимости**: Структуры не могут быть объявлены как private или protected. Они всегда public.
4. **Использование**: Структуры подходят для небольших, простых объектов, которые обычно не подвергаются частым изменениям. Классы обычно используются для создания более сложных объектов с состоянием и поведением.
5. Определение перечисления

Перечисление (Enumeration) – это: набор (последовательность) констант, которые образуют тип данных (значение каждой константы задается именем);

1. Общая форма объявления перечисления (объявление перечисления)

enum название\_перечисления

{ // значения перечисления значение1,

значение2,

.......

значениеN

}

1. Понятие наследования в ООП

Наследование в объектно-ориентированном программировании (ООП) - это механизм, позволяющий создавать новые классы на основе существующих (родительских) классов. При наследовании дочерний класс (подкласс) автоматически получает свойства и методы родительского класса (суперкласса), а также может расширить или изменить их функциональность.

1. Особенности базового и производного классов

Особенности базового и производного классов в объектно-ориентированном программировании:

1. **Базовый класс (родительский)**:
   * Определяет общую функциональность и свойства, которые наследуются производными классами.
   * Может быть абстрактным, то есть содержать абстрактные методы, которые требуют реализации в производных классах.
2. **Производный класс (дочерний)**:
   * Наследует свойства и методы базового класса.
   * Может расширять функциональность базового класса, добавляя новые методы и свойства.
   * Может переопределять методы базового класса, чтобы изменить их поведение.
3. Синтаксис объявления производного класса (потомка)

class DerivedClassName : BaseClassName

{

// Дополнительные члены класса

}

1. Описание конструктора производного класса

class DerivedClassName : BaseClassName

{

public DerivedClassName(/\* Параметры конструктора \*/) : base(/\* Аргументы для вызова конструктора базового класса \*/)

{

// Инициализация дополнительных членов производного класса

}

}

1. Понятие виртуального метода

Виртуальный метод в объектно-ориентированном программировании (ООП) - это метод, который может быть переопределен (перекрыт) в производном классе.

1. Общая форма объявления виртуального метода

class BaseClass

{

public virtual void VirtualMethod()

{

// Тело виртуального метода

}

}

1. Понятие переопределения метода

Переопределение метода в объектно-ориентированном программировании (ООП) - это процесс изменения реализации метода, который уже определен в базовом классе, в производном классе.

1. Виды наследования

 **Одиночное наследование**: Класс наследует функциональность только от одного базового класса.

 **Множественное наследование интерфейсов**: Класс может реализовать (наследовать) несколько интерфейсов, что позволяет ему обеспечивать реализацию множества контрактов.

1. Понятие абстрактного класса

Абстрактный класс - это специальный класс в C #, который не может быть создан, т.е. вы не можете создавать объекты абстрактного класса.

1. Особенности абстрактного класса

Абстрактный класс имеет следующие особенности:  
 - нельзя создавать экземпляры (объекты) абстрактного класса;  
 - абстрактный класс может содержать как абстрактные методы/свойства, так и обычные;   
- в классе наследнике должны быть реализованы все абстрактные методы и свойства, объявленные в базовом классе

1. Общая форма объявления абстрактного класса

[модификатор доступа] abstract class [Имя\_класса] { //тело (абстрактные и неабстрактные элементы) }

1. Понятие абстрактного метода

Абстрактный метод – это метод, который не имеет своей реализации в базовом классе, и должен быть реализован в классе-наследнике.

1. Отличия между абстрактным и виртуальным методами

1) виртуальный метод может иметь свою реализацию в базовом классе, абстрактный – нет (тело пустое);

2) абстрактный метод должен быть реализован в классе-наследнике, виртуальный метод переопределять необязательно

1. Понятие обобщения

Обобщение в программировании - это механизм, который позволяет создавать обобщенные (generic) типы и методы, которые могут работать с различными типами данных, не привязываясь к конкретному типу на этапе компиляции.

1. Понятие делегата

Делегат - это тип, который представляет собой ссылки на методы с определенным списком параметров и возвращаемым типом

1. Описание делегата

public delegate int PerformCalc (int x, int y);

1. Алгоритм создания делегата

Алгоритм создания делегата в C#:

1. **Определите делегат**: Определите тип делегата, который будет представлять метод, который вы хотите передать.

csharp

Копировать код

delegate void MyDelegate(int x);

1. **Создайте экземпляр делегата**: Создайте экземпляр делегата, указывая на метод, который вы хотите вызвать.

csharp

Копировать код

MyDelegate del = new MyDelegate(SomeMethod);

1. **Создайте метод, который соответствует делегату**: Создайте метод, который имеет сигнатуру, совместимую с делегатом.

csharp

Копировать код

static void SomeMethod(int x)

{

Console.WriteLine($"SomeMethod was called with argument {x}");

}

1. **Вызовите делегат**: Вызовите делегат, передав нужные аргументы.

csharp

Копировать код

del(10); // Вызовет SomeMethod(10)

В результате выполнения этих шагов у вас будет экземпляр делегата, который может быть использован для вызова метода, определенного в его сигнатуре.