МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

"Казанский (Приволжский) федеральный университет"

Набережночелнинский институт (филиал)

Отделение информационных технологий и энергетических систем

Кафедра информационных систем

ПРОГРАММИРОВАНИЕ

Материалы презентаций курса

Мингалеева Лейсэн Башировна

Тазмеев Алмаз Харисович

Галиуллин Ленар Айратович

Казань – 2018

Текст презентации к лекции №1

Начало работы с C#

Обзор модуля:

- Введение в C#
- Особенности языка
- Среду Visual Studio 2005
- Элементы Microsoft Visual Studio 2005 IDE

Занятие 1 - Введение в C#

На этом первом занятии, **Введение в C#**, вы научитесь:

- Формулировать ловушки C/C++.
- Определять C# как новый язык.
- Описывать цель языка C#.
- Описывать .NET Framework.
- Перечислять и объяснять компоненты .NET Framework.
- Описывать архитектуру .NET Framework.
- Перечислять другие компоненты .NET Framework.
- Объяснять Общеязыковую Исполняющую Среду (CLR) и Промежуточный Язык Microsoft (MSIL).

Ловушки C/C++

Некоторые недостатки:

- Сложен в изучении для начинающих
- Нельзя проверить код до компиляции
- Сложность отладки без дорогих отладочных инструментов
- Требует много времени на разработку
- Сложное подключение к базам данных
- Проблемы при модификации данных
- Сложность при реализации
- Не содержит никаких структур

Необходимость нового языка

C# был разработан для предоставления следующих преимуществ:

- Создания очень простых и мощных инструментов для разработки взаимодействующих, масштабируемых и надежных приложений
- Создания полностью объектно-ориентированной архитектуры
- Поддержки мощной компонентно-ориентированной разработки
- Получения доступа ко многим возможностям, доступным только в C++, при сохранении простого в использовании быстрого инструмента разработки, например Visual Basic
- Обеспечения легкого знакомства с языком программистов, переходящих с C или C++
- Написания приложений для настольных ПК и мобильных устройств

Цели языка C#

- Microsoft.NET ранее известен как Next Generation Windows Services (NGWS)- ”Следующее Поколение Сервисов Windows”

- Это абсолютно новая платформа для разработки нового поколения Windows/Web приложений.

- C# это объектно-ориентированный язык производный от C и C++.

- Представляет собой простой, эффективный, производительный, объектно-ориентированный язык.

Основы .NET Framework

.NET Framework разработан для:

- Обеспечения единообразного объектно-ориентированного программного окружения
- Минимизации конфликтов при развертывании приложений и управлении версиями за счет предоставления выполняющего код окружения
- Обеспечения безопасного выполнения кода за счет передоставления выполняющего код окружения
- Предоставления последовательного опыта разработки через разные типы приложений, например Windows-приложения и Web-приложения

Компоненты .NET Framework

.NET Framework содержит два основных компонента:

- CLR (Common Language Runtime) – “Общеязыковая Исполняющая Среда”
- FCL (Framework Class Library) - Библиотека Классов .NET Framework

Общеязыковая исполняющая среда -

- основа .NET Framework.
- выполняет различные функции, например:
  - Управление памятью
  - Выполнение кода
  - Обработка ошибок
  - Проверка кода на безопасность
  - Сборка мусора

Компоненты .NET Framework

Организация .NET Framework

- Это многоязыковое и многоплатформенное окружение для сборки, развертывания и выполнения приложений.

- Оно позволяет программисту разрабатывать приложения для различных платформ, таких как мобильные устройства, персональные компьютеры, смартфоны и т.д..

- Архитектура .NET Framework включает в себя следующее:
  - Языки, включенные в .NET Framework
  - CLR (Common Language Runtime) – “Общеязыковая Исполняющая Среда”
  - FCL (Framework Class Library) - Библиотека Классов .NET Framework

Организация .NET Framework

Организация.NET Framework

Использование .NET Framework

- Программисты, разрабатывающие приложения, используют один из языков, поддерживаемых .NET.

- Эти приложения используют базовые библиотеки классов предоставленные Библиотекой Классов .NET

- Следующая команда может быть использована для вывода текстового сообщения на экран:

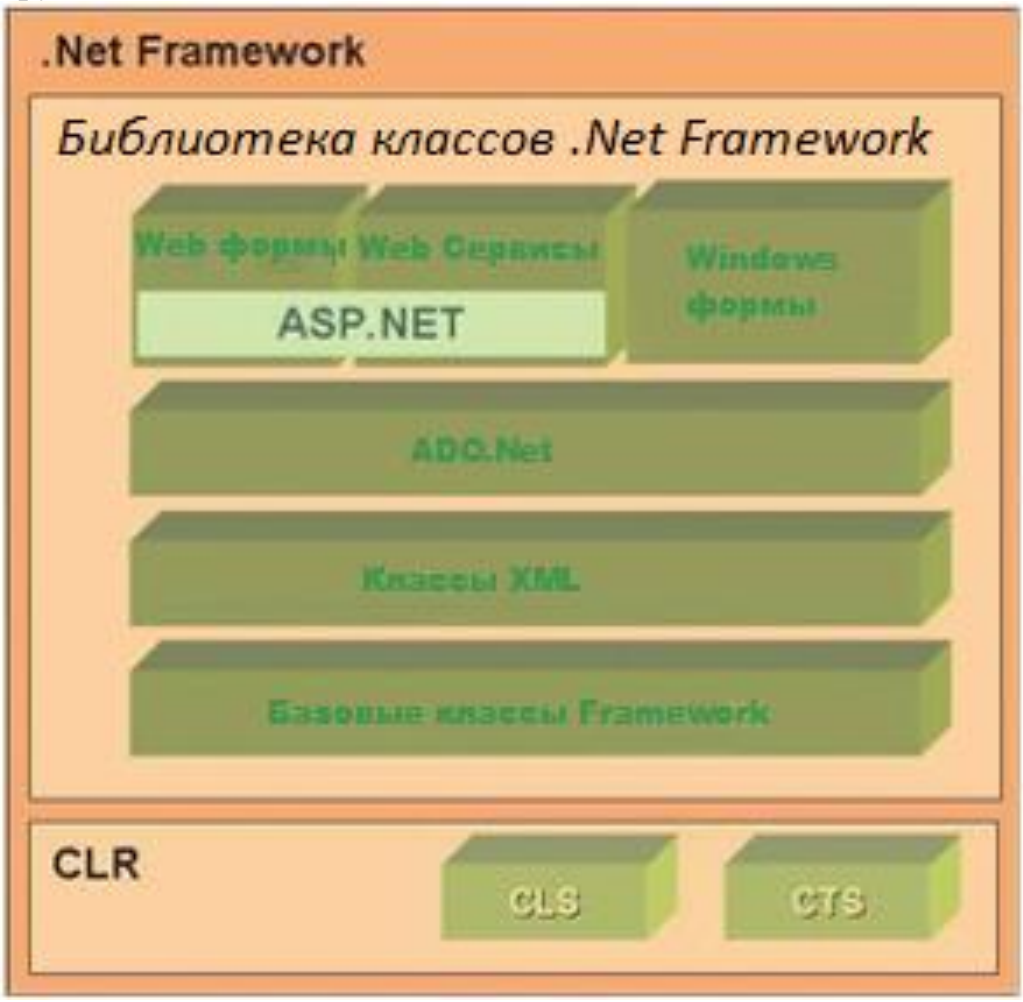
```
System.Console.WriteLine("Архитектура .NET");
```

Другие компоненты .NET Framework

- Общеязыковая Исполняющая Среда (CLR) и Библиотека Классов .NET Framework (FCL) - основные компоненты .NET Framework.
- Остальные важные компоненты описаны ниже:
  Web Forms
  Web-сервисы
  Windows Forms
  ASP.NET
  ADO.NET
  Класс XML

Базовые классы инфраструктуры
Общая спецификация для языков программирования (CLS)
Общая система типов (CTS)

Другие компоненты .NET Framework



Другие компоненты .NET Framework

- **Web Forms** - предоставляет набор классов для разработки форм web-страниц, похожих на HTML формы.

- **Web-сервисы** - включает набор классов для разработки приложений, которые могут получать доступ используя стандартный стек протоколов.

- **Windows Forms** - предоставляет набор классов для разработки форм windows-приложений

Другие компоненты .NET Framework

- **ASP.NET** - предоставляет набор классов для разработки web-приложений.

- **ADO.NET -** предоставляет классы для взаимодействия с базами данных.

- **Класс XML –** позволяет манипулировать XML, проводить поиск и преобразование.

Другие компоненты .NET Framework

- **Базовые классы инфраструктуры -** обеспечивают базовую функциональность, такую как ввод/вывод, обработку строк, управление безопасностью, сетевое взаимодействие и т.д.
- **Общая спецификация для языков программирования (CLS) -** набор правил, которым должен следовать любой язык .NET, для создания приложений, взаимодействующих с приложениями на других языках.
- **Общая система типов (CTS) –** описывает, как типы данных объявляются, используются и управляются во время выполнения и облегчает использование типов различных языков.

Язык промежуточного уровня Microsoft (MSIL).

- Каждый язык программирования в .NET обычно имеет свой собственный компилятор и среду выполнения.

- Когда код, написанный на совместимом с .NET языке, например C# или VB, компилируется, получается код в форме MSIL.

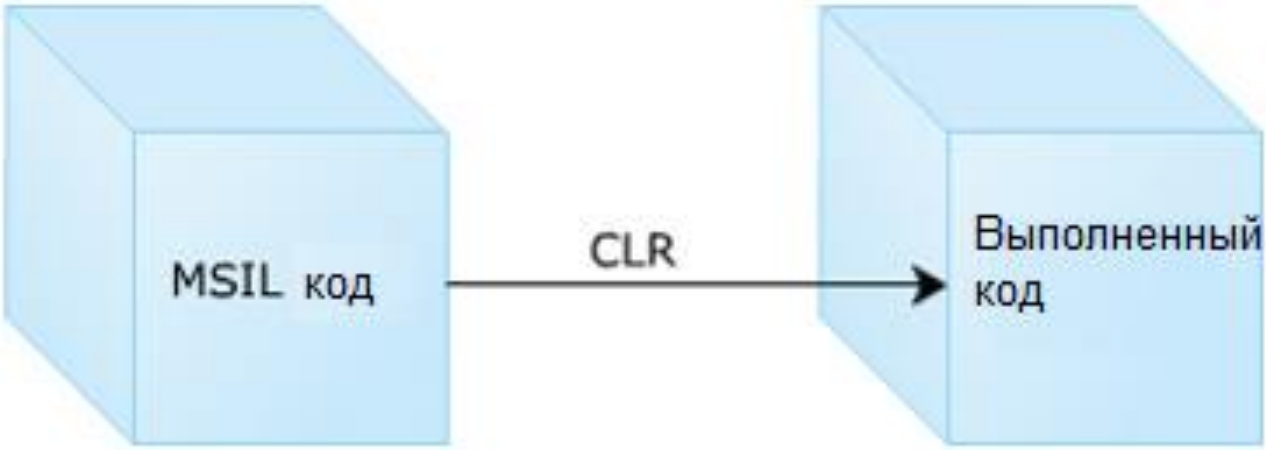
- MSIL формирует набор инструкций, которые указывают, как код должен быть выполнен.

Язык промежуточного уровня Microsoft (MSIL).

Общеязыковая исполняющая среда (CLR)

- CLR это виртуальная машина компонентов .NET, которая используется для преобразования кода MSIL в машинные инструкции.

- Это происходит во время работы компилятора Just-In-Time (JIT), присутствующего в CLR.



Занятие 2 – Возможности языка

На втором занятии, **Возможности языка**, вы научитесь:

- Перечислять базовые возможности C#.
- Определять общие приложения C#.
- Формулировать преимущества C#.
- Определять управление памятью и сборщик мусора.

Базовые возможности C#

- C# это язык программирования, созданный для разработки широкого спектра приложений, выполняемых в .NET Framework.

- Некоторые ключевые особенности:
  - Объектно-ориентированное программирование
  - Проверка безопасности типов
  - Сборщик мусора
  - Стандартизация европейской ассоциацией изготовителей компьютеров (ECMA)
  - Обобщенные типы и методы

Базовые возможности C#

Приложения C#

- C# может использоваться в различных приложениях, например таких как:
  - Игры
  - Крупные промышленные приложения
  - Мобильные приложения для карманных персональных компьютеров (PC), цифровых секретарей(PDA) и сотовых телефонов
  - Простые изолированные настольные приложения, например система управления библиотекой, генератор сводной экзаменационной ведомости студентов и другие
  - Комплексные распределенные приложения, которые охватывают города или

целые страны

Преимущества C#

- **Поддержка нескольких языков** – Код, написанный на любом языке .NET, может быть легко интегрирован в приложения C#.
- **Общие протоколы Интернет** - .NET предлагает всестороннюю поддержку XML, который является предпочтительным выбором для форматирования информации в Интернет.
- **Простое развертывание** - Развертывание приложений C# упрощено благодаря концепции сборок.
- **Документация XML** - Комментарии могут быть переведены в формат XML и затем использованы для документирования кода.

Преимущества C#

- C# предпочтительнее, чем C++, поскольку он проще и удобнее в использовании.

- Преимущества C#:
  - Поддержка нескольких языков
  - Общие протоколы Интернет
  - Простое развертывание
  - Документация XML

Управление памятью

- В таких языках программирования как C и C++, выделение и освобождение памяти выполняется вручную.

- C# предоставляет возможность выделять и освобождать память, используя автоматическое управление памятью.

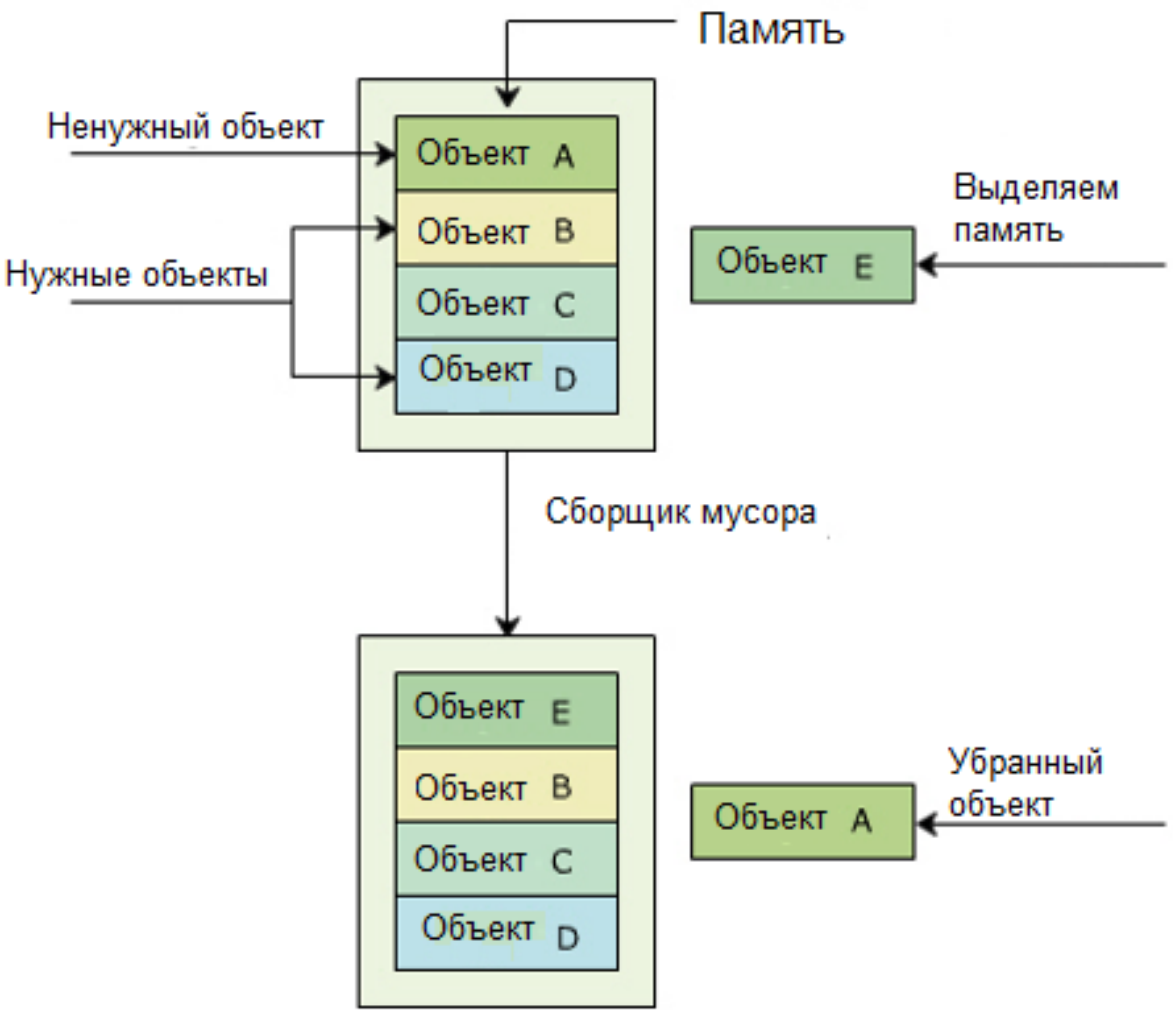
- Автоматическое управление памятью улучшает качество кода, повышает производительность и продуктивность.

Сборщик мусора

- Процесс автоматического выделения и освобождения памяти с использованием автоматического управления памятью производится с помощью сборщика мусора.

- Автоматически очищает память от объектов, которые больше используются.

Сборщик мусора



Занятие 3 - Среда Visual Studio 2005

На третьем занятии, **Среда Visual Studio 2005**, вы научитесь:

- Описывать продукт Microsoft Visual Studio 2005.
- Перечислять различные редакции Visual Studio 2005.
- Перечислять языки, поддерживаемые Visual Studio 2005.
- Перечислять и объяснять возможности Visual Studio 2005.

Введение в Visual Studio 2005

- Это полный набор средств разработки для создания высокопроизводительных настольных приложений, XML Web-сервисов, мобильных приложений и приложений ASP Web.
- Также используется для упрощения командного проектирования, разработки и развертывания промышленных решений.
- Основные преимущества:
  - Увеличивает продуктивность разработчика
  - Разработка приложений для Microsoft .NET Framework 2.0
  - Разработка приложений для портативных устройств использующих Microsoft .NET Compact Framework 2.0

Редакции Visual Studio 2005

- Интегрированная среда разработки (IDE) Microsoft Visual Studio это результат интенсивных исследований команды Microsoft.

- Visual Studio 2005 имеет четыре редакции:
  - Team System Edition
  - Professional Edition
  - Standard Edition
  - Express Edition

Редакции Visual Studio 2005

Редакции Visual Studio 2005

Языки в Visual Studio 2005

- Visual Studio 2005 поддерживает несколько языков программирования.

- Языки, поддерживаемые Visual Studio 2005:
  - Visual Basic
  - Visual C++
  - Visual C#
  - Visual J#

Возможности Visual Studio 2005

- Visual Studio 2005 предоставляет новые возможности, например:
  - Всеобъемлющая инструментальная платформа
  - Снижение сложности разработки
  - Заметки при редактировании
  - Шаблоны кода
  - Автоматическое восстановление
  - Интеллектуальное восприятие
  - Рефакторинг

Возможности Visual Studio 2005

- **Всеобъемлющая инструментальная платформа -** Разработчики любого уровня подготовки могут использовать инструменты разработки, которые способствуют развитию опыта для их индивидуальных нужд.
- **Снижение сложности разработки -** Позволяет предоставлять клиентам более простые решения широкого спектра, базирующиеся на .NET Framework, включая Windows, Office, Web и мобильные приложения.
- **Заметки при редактировании -** Предоставляет визуальные сведения о сделанных, но не сохраненных изменениях и изменениях, сделанных во время текущей сессии, которые не были сохранены на диск.

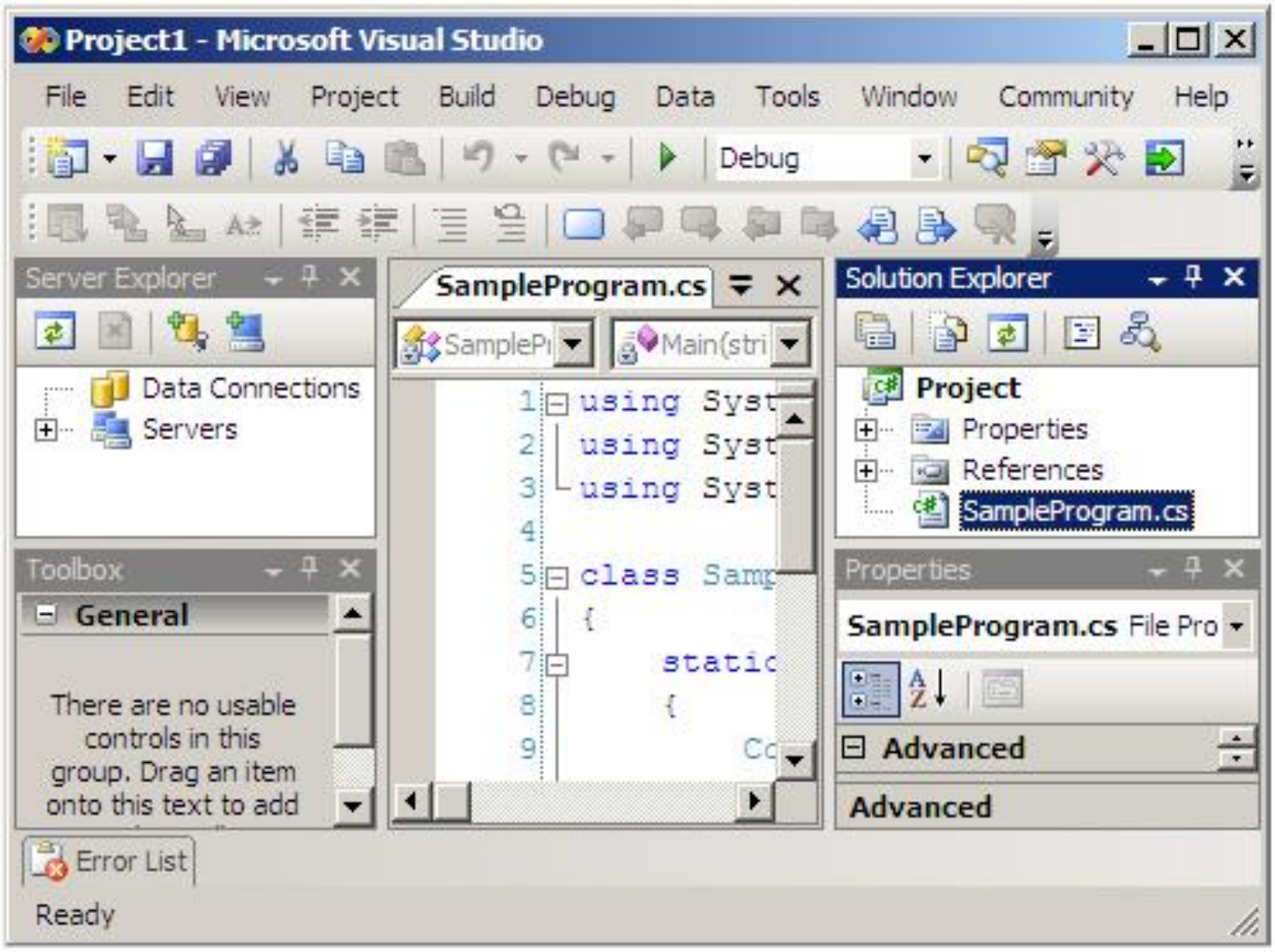
Возможности Visual Studio 2005

- **Шаблоны кода -** Небольшие модули исходного кода C#, которые разработчик может быстро использовать с помощью горячих клавиш.
- **Автоматическое восстановление -** Автоматическое регулярное сохранение результатов работы минимизирует потери информации при неожиданном закрытии несохраненных файлов.
- **Интеллектуальное восприятие -** Делает процесс ввода кода более эффективным.

- **Рефакторинг -** Позволяет разработчику автоматизировать часто выполняемые задачи при реструктуризации кода.

Занятие 4 - Элементы Microsoft Visual Studio 2005 IDE
На последнем занятии, **Элементы Microsoft Visual Studio 2005 IDE**, вы научитесь:
- Узнавать ключевые элементы Visual Studio 2005 IDE.
- Описывать процесс компиляции и выполнения программы C# (из командной строки и из IDE).

Ключевые элементы



Ключевые элементы

- Microsoft Visual Studio 2005 IDE - это набор инструментов для разработки, доступных через общий пользовательский интерфейс.

- Ключевые элементы Visual Studio 2005 IDE:
  - Обозреватель решений (Solution Explorer)
  - Редактор кода (Code Editor)
  - Окно свойств (Properties Window)
  - Динамическая помощь (Dynamic Help)

Ключевые элементы

- **Обозреватель решений (Solution Explorer) -** Предоставляет организованное отображение проектов и файлов.
- **Редактор кода (Code Editor) -** Используется для написания, отображения и редактирования форм, событий и кода методов.
- **Окно свойств (Properties Window) -** Используется для просмотра и редактирования свойств во время редактирования, и событий для выбранных объектов.
- **Динамическая помощь (Dynamic Help) -** Предоставляет список тем, специфичных для области IDE, в которой вы работаете, или для задач, которые вы выполняете.

Команда “csc”

- Команда компилятора C# (C Sharp Compiler, читается «си шарп»), (csc) может быть использована для компилирования C# программ.

- Для компиляции и выполнения программы выполните следующие шаги:
  - Создайте новый проект
  - Скомпилируйте C# программу
  - Выполните программу

Команда “csc”

- **Создайте новый проект**
  1. Запустите Visual Studio 2005.
  2. Выберите "New Project" из меню "File".
  3. В диалоговом окне "New Project" выберите "Visual C#" из "Project types" и "Console Application" из секции "Templates".
  4. Укажите название и место размещения проекта.

Команда “csc”

- **Компиляция C# программы -** C# программа может быть скомпилирована с использованием следующего синтаксиса:

Синтаксис

```
csc <file.cs>
```

- Пример:
-

```
csc SampleProgram.cs
```

где,

`SampleProgram`: имя программы для компиляции.

Команда “csc”

- **Выполнить программу -** Наберите имя_файла.exe в командной строке.

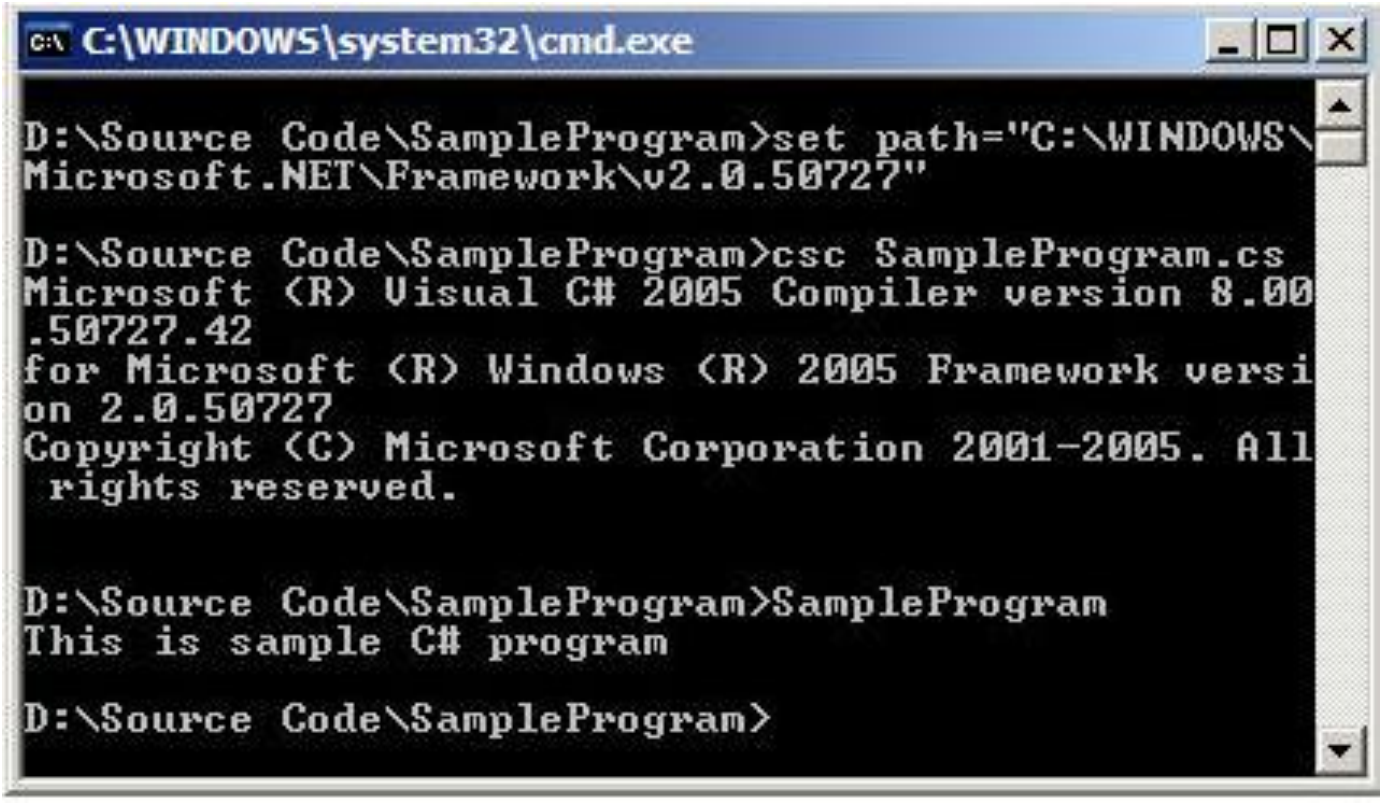
- Пример:

```
D:\Source Code\SampleProgram>SampleProgram
```

где,

`D:\Source Code\SampleProgram`: путь к программе.

Команда “csc”



Сборка и выполнение

- IDE предоставляет необходимую поддержку для компиляции и выполнения C#

программ.

- Необходимые шаги:
  - Компиляция C# программы
  - Выполнение программы

Сборка и выполнение

- **Компиляция C# программы**

Выберите «Построить решение (Build) <имя приложения>" из меню “Построить (Build)".

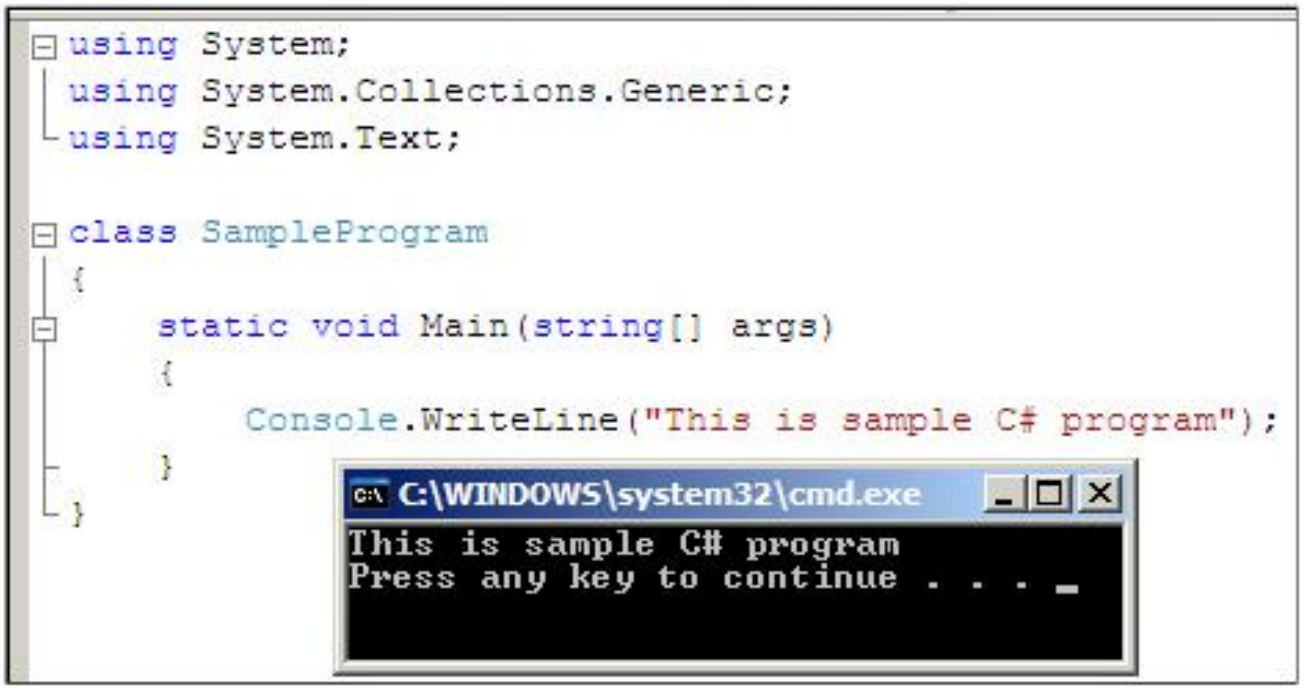
Это действие создаст исполняемый файл (.exe).

- **Выполнение программы**

В меню "Отладка (Debug)" выберите "Выполнить без отладки (Start Without Debugging)".

Вывод программы будет отображен на экране.

Сборка и выполнение



Заключение

- **Введение в C#**
  - C# это объектно-ориентированный язык производный от C и C++.
- **Особенности языка**
  - Ключевая особенность – это его объектно-ориентированность.
  - Он также поддерживает такие возможности, как проверка безопасности типов, сборка мусора, ECMA стандартизация и обобщения.

- **Среда Visual Studio 2005**
  - Это полный набор средств разработки для создания высокопроизводительных настольных приложений , XML Web-сервисов, мобильных приложений и приложений ASP Web.
- **Элементы Microsoft Visual Studio 2005 IDE**
  - Это набор инструментов разработки, доступных через общий пользовательский интерфейс.

Текст презентации к лекции №2

Переменные и типы данных

Обзор модуля

Вы изучите:

- Переменные и типы данных в C#
- Комментарии и XML-документацию
- Константы и литералы
- Ключевые слова и Escape-последовательности
- Ввод и вывод

Занятие 1 - Переменные и типы данных в C#

На первом занятии, Переменные и типы данных в C#, вы научитесь:

- Описывать переменные и их цели.
- Описывать типы данных и их цели.
- Определять базовые типы данных в C#.
- Описывать ссылочные типы данных.
- Объяснять правила именования переменных.
- Объявлять и использовать переменные.

Определение

- **Переменная** - это сущность, значение которой может изменяться. Например, возраст студента, зарплата сотрудника.

- В C# **переменные -** это область компьютерной памяти, которая идентифицируется уникальным именем и используется для хранения значений.
- Они базируются на типах данных, которые необходимы для хранения (переменные могут быть различных типов).

Использование переменных

- В C# память под переменные выделяется в момент их создания.
- Переменной в момент создания присваивается имя, которое уникально идентифицирует переменную в ее области видимости.
- Вы можете инициализировать переменную в момент создания или позднее.
- Переменные позволяют вам отслеживать данные. Когда вы обращаетесь к переменной, вы на самом деле обращаетесь к значению, хранящемуся в этой переменной.

Синтаксис - Объявление

```
<тип данных> <имя переменной>;
```

Синтаксис - Инициализация

```
<имя переменной> = <значение>;
```

где,
= : оператор присваивания, используемый для установки значения.

Типы данных

- Компилятор должен знать, какой тип данных хранит конкретная переменная.
- Это позволяет переменным хранить значения соответствующих типов данных.
- В языке программирования C# типы данных делятся на две категории. Это:
  - Типы-значения
  - Ссылочные типы

Типы значения

- Хранят актуальное значение в стеке
- Значения могут быть любого встроенного или определенного пользователем типа данных

- Значения встроенных типов данных → int, float, double, char и bool

Ссылочные типы

- Хранят адреса размещения в памяти других переменных
- Значения могут принадлежать любому встроенному или пользовательскому типу данных
- Большинство определенных пользователем типов, например, классы, являются ссылочными типами

Тип данных	Размер	Диапазон
byte	Беззнаковое 8-битное целое	От 0 до 255
short	знаковое 16-битное целое	От -32,768 до 32,767
int	знаковое 32-битное целое	От -2,147,483,648 до 2,147,483,647
long	знаковое 64-битное целое	От – 9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
float	32-битное вещественное число с точностью 7 знаков	От ±1.5e−45 до ±3.4e38
double	64-битное вещественное число с точностью 15-16 знаков	От ±5.0e−324 до ±1.7e308
decimal	128-битное вещественное число с точностью 28-29 знаков	От ±1.0 × 10e−28 до ±7.9 × 10e28
char	16-битный символ Unicode	От U+0000 до U+ffff
bool	Хранит значения true или false	true или false

Символы Unicode

- 16-битные символы (с префиксом “U”) используются для отображения различных языков, использующихся в мире

Знаковые целые

- Представляют и положительные, и отрицательные числа

Представление символьных типов данных и типов с плавающей точкой

- Переменные типов с плавающей точкой всегда заканчиваются символом “F” или “f” и всегда должны быть заключены в одинарные кавычки

Классификация

- Ссылочные типы данных хранят в памяти ссылки на другие переменные, которые содержат актуальные значения.
- Классификация ссылочных типов данных:
  - Объект
  - Строка
  - Класс
  - Делегат
  - Интерфейс
  - Массив

Объект

- Это встроенный ссылочный тип данных.
- Это базовый класс для всех предопределенных и пользовательских типов данных.

Строка

- Это встроенный ссылочный тип данных.
- Представляет строку символов Unicode.
- Позволяет присваивать и изменять строковые значения.
- Однажды созданная строка не может быть модифицирована.

Класс

- Это определенная пользователем структура, содержащая переменные и методы.

Делегат

- Это определенный пользователем тип, содержащий ссылку на один или более методов.

Интерфейс

- Это тип определенного пользователем класса, который используется для сложного наследования.

Массив

- Это определенная пользователем структура, содержащая значения одного типа данных, например, оценки студентов.

Правила

Правила объявления переменных:

- Имя переменной может начинаться с буквы в верхнем или нижнем регистре. Имя может содержать буквы, цифры и символ подчеркивания (_).
- Первый символ в имени переменой должен быть буквой и не может быть цифрой. Подчеркивание также допустимый первый символ, но он не рекомендуется для начала имени.
- Язык C# чувствителен к регистру; таким образом переменные с именами count и Count - две разные переменные.
- Ключевые слова C# не могут быть использованы в качестве имен переменных. Если вам необходимо использовать ключевые слова C#, используйте символ ‘@’ как префикс.

Проверка

Имя переменной	Допустимо/недопустимо
Employee	Допустимо
student	Допустимо
_Name	Допустимо
Emp_Name	Допустимо

@goto	Допустимо
static	Недопустимо, поскольку слово ключевое
4myclass	Недопустимо, поскольку имя переменной не может начинаться с цифры
Student&Faculty	Недопустимо, поскольку имя переменной не может содержать специальный символ &

Объявление

- В C# вы можете одновремено объявлять несколько переменных, так же как объявляете одиночную переменную.
- После объявления переменной вам необходимо присвоить ей значение.
- Присвоение значения переменой называется инициализацией.
- Присвоить переменной значение вы можете в момент создания или позднее.

Синтаксис – Объявление одной переменой

```
<тип данных> <имя переменной> = <значение1>;
```

Синтаксис – Объявление нескольких переменных

```
<тип данных> <имя переменной1>, <имя переменной2 >,..,   <имя переменнойN>;
```

Синтаксис – Объявление нескольких переменных и инициализация

```
<тип данных> <имя переменной1> = <значение>, <имя переменной2> = <значение2>;
```

Пример:

```
bool b1 = true;
short b2 = 19;
int i;
string s = “Иван”;
float f;
i =140000;
fl = 14.5f;
Console.WriteLine(“b1 = {0}”, b1);
Console.WriteLine(“b2 = ” + b2);
Console.WriteLine(“i = ” + i);
Console.WriteLine(“s = ” + s);
Console.WriteLine(“f = ” + f);
```

Занятие 2 - Комментарии и XML-документация

На втором занятии, **Комментарии и XML-документация**, вы научитесь:

- Описывать комментарии в C# приложениях.
- Объяснять XML-документацию исходного кода.

Определение 1-2

Комментарии

- Предоставляют информацию о части кода.
- Делают программу более читаемой.
- Помогают программисту объяснить цель использования редких переменных или методов.
- Идентифицируются с помощью специальных символов.
- Игнорируются компилятором при выполнении программы
- C# поддерживает три типа комментариев:
  - Однострочные комментарии
  - Многострочные комментарии
  - XML-комментарии

Определение 2-2

Однострочные комментарии

- Начитаются с двух слэшей (//)

```
// Это блок кода с двумя числами
int Sum = 4 + 3;
или
// Это блок кода складывает два числа и помещает
// результат в переменную doSum
int Sum = 4 + 3;
или
int Sum = 4 + 3; // Сложение двух чисел
```

Многострочные комментарии

- Многострочные комментарии начинаются со слэша, за которым следует звездочка (/*) и заканчиваются звездочкой, за которой следует слэш (*/).

- Многострочные комментарии могут быть вставлены между начальным и конечным символами.

```
/* Это блок кода, который
   перемножает два числа, затем
   делит результат на 2 и
   выводит частное */
int Mult = 5 * 20;
int Div = Mult / 2;
Console.WriteLine(“Частное:” + Div)
```

XML-комментарии

- XML-комментарии начинаются с трех слэшей (///).
- В отличии от одно- и многострочных комментариев, XML-комментарии должны заключаться в XML-тэги. Вам необходимо создать XML-тэг для вставки XML комментария.

```
/// <summary>
/// Вы в тэге XML называемом summary.
/// </summary>
```

Документация XML

- В C# XML-документ содержит все созданные XML-комментарии. Этот документ используется, когда несколько программистов хотят увидеть информацию о программе.
- Например, один программист хочет понять технические детали кода, а другой хочет увидеть все переменные, использованные в коде. В этом случае вы можете создать XML-документ, который будет содержать всю необходимую информацию.
- Для создания XML-документа вы должны использовать окно Visual Studio 2005 Command Prompt.

```
class XMLcomments
    {
        /// <summury>
        /// Это XML комментарий и он может быть извлечен в XML файл.
        /// </summary>
        static void Main(string[] args)
        {
            Console.WriteLine("Данная программа иллюстрирует XML
комментарии");
        }
```

```
}
```

Синтаксис

```
csc /doc: <Имя файла XML.xml> <Имя файла C#.cs>
```

Предопределенные XML тэги

- XML-комментарии вставляются в XML-тэги.
- Эти тэги могут быть предопределенными или определенными пользователем.

Предопределенные теги	Описание
<c>	Устанавливает для текста шрифт кода
<code>	Устанавливает одну или более строк исходного кода или вывода программы.
<example>	Обозначает пример.
<param>	Описывает параметр метода или конструктора.
<returns>	Описывает возвращаемое методом значение.
<summary>	Резюмирует общую информацию о коде.

Занятие 3 - Константы и литералы

На третьем занятии, Константы и литералы, вы научитесь:

- Описывать константы в C#.
- Перечислять различные типы литералов.

Необходимость констант

- Константы в C# - это фиксированные значения, присвоенные идентификаторам, которые не меняются при выполнении кода.
- Константы определяются тогда, когда значения должны быть фиксированными и повторно используемыми или для предотвращения их модификации.

Константы

- В C# константы могут быть определены для всех типов данных.
- Вы должны инициализировать константы в момент их создания.
- Константы объявляются для типов значений, а не для ссылочных типов.
- Для объявления идентификатора как константы, используется ключевое слово const в объявлении идентификатора. Компилятор идентифицирует константы при компиляции по ключевому слову const.

Синтаксис

```
const <тип данных> <имя идентификатора> = <значение>;
```

где,

const: Ключевое слово, означающее, что идентификатор объявляется как константа.

Пример

```
const float _pi = 3.14F;
float r = 5;
float a = _pi * r * r;
Console.WriteLine("Площадь круга: " + a);
```

Использование литералов 1-4

- Литерал - это статическое значение, присвоенное переменной или константе.
- Литералы могут быть определены для любого типа данных C#.
- Числовые литералы могут содержать суффикс в виде символа алфавита, определяющего тип данных литерала. Литералы могут быть в верхнем или нижнем регистре. Например, string bookName = “Csharp”.

В C# есть шесть типов литералов. Это:

- Логические литералы
- Целые литералы
- Вещественные литералы
- Символьные литералы
- Строковые литералы
- Null Literal

Использование литералов 2-4

Логические литералы

- Логические литералы могут иметь два значения: true или false.

Пример

```
bool val = true;
```

где,

true: логический литерал, присвоенный переменной val.

Целые литералы

- Целые литералы могут присваиваться типам данных int, uint, long или ulong. Суффиксы для целых литералов включают U, L, UL или LU. U обозначает uint или ulong, L обозначает long. UL и LU обозначает ulong.

Пример

```
long val = 53L;
```

где,

53L: целый литерал, присвоенный переменной val.

Вещественные литералы

- Вещественные литералы присваиваются типам данных float, double (по умолчанию), decimal. На это указывает символ-суффикс, добавляемый после присваиваемого значения. Вещественный литерал может заканчиваться на F, D или M. F обозначает float, D обозначает double и M обозначает decimal.

Пример

```
float val = 1.66F;
```

где,

1.66F: вещественный литерал, присвоенный переменной val.

Символьные литералы

- Символьные литералы присваиваются типу данных char. Символьный литерал всегда заключается в одинарные кавычки.

Пример

```
char val = 'A';
```

где,

A: символьный литерал, присвоенный переменной val.

Строковые литералы

- Есть два типа строковых литералов в C#, обычные и дословные.
- Обычный строковый литерал - это стандартная строка.
- Дословный строковый литерал похож на обычный строковый литерал, но имеет префикс @.
- Строковый литерал всегда заключается в двойные кавычки.

Пример

```
string s = "@gmail.com";
```

где,

@gmail.com: дословный строковый литерал.

Null Literal

- Null-литерал имеет только одно значение - null.

Пример

```
string s = null;
```

где,

null: Определяет s, не ссылающийся ни на один объект (ссылку).

Занятие 4 - Ключевые слова и escape-последовательности

На четвертом занятии, **Ключевые слова и escape-последовательности**, вы научитесь:

- перечислять ключевые слова в C#.
- Перечислять и объяснять символы escape-последовательностей.

Ключевые слова

- Ключевые слова - это зарезервированные слова, которые обрабатываются компилятором отдельно.
- Ключевые слова имеют предопределенное значение для компилятора и поэтому не могут быть созданы или модифицированы.
- Например, int - это ключевое слово, которое указывает, что переменная имеет тип данных integer.

- Ключевые слова не могут быть использованы в именах переменных, методов или классов (за исключением случаев использования префикса в виде символа “@”).

abstract	bool	break	byte	case	catch
char	class	const	continue	default	double
enum	else	false	finally	float	for
foreach	goto	if	int	interface	long
namespace	new	public	private	protected	return
sbyte	short	static	string	struct	switch
throw	true	try	ushort	void	while

Необходимость в символах escape-последовательностей

- Рассмотрим фонд заработной платы в какой-либо организации.
- Одна из его функций - показывать месячную зарплату, при этом каждое значение должно выводиться в новой строке.
- Программист хочет написать код, который всегда печатает зарплату в новой строке вне зависимости от длины строки, представляющей размер заработной платы.
- Для этого используются escape-последовательности.

Определение

- Символ escape-последовательности - это специальный символ, следующий за символом обратного слэша (\).
- Символы escape-последовательностей используются для реализации специальных непечатных символов, например, новой строки, одиночного пробела или возврата каретки.
- Непечатные символы используются при форматированном выводе для повышения удобочитаемости.
- Символ обратного слэша сообщает компилятору, что следующий символ отмечен как непечатный.
- Например, \n используется для добавления новой строки подобно клавише Enter на клавиатуре.
- В C# символы escape-последовательности должны всегда заключаться в двойные кавычки.

Символы escape-последовательностей в C#

В C# существует много символов escape-последовательностей, которые используются для различных видов форматирования.

Символы	Непечатные символы
\’	Одиночная кавычка, необходима для символьных литералов
\”	Двойная кавычка, необходима для строковых литералов
\\	Обратный слэш, необходим для строковых литералов
\0	Символ Unicode 0.
\a	Предупреждение.
\b	Backspace.
\f	Перевод страницы.
\n	Новая строка.
\r	Возврат каретки.
\t	Горизонтальная табуляция.
\v	Вертикальная табуляция.
\xhh	Подбирает ASCII-символ, используя шестнадцатеричное представление (две цифры). Например, \x61 представляет символ ‘a’.

Пример

```
string str = “\u0048\u0065\u006C\u006C\u006F”;

Console.Write(“\t” + str + “!\n”);

Console.WriteLine(“Иван\u0020\“2007\” ”);
```

Вывод

```
Hello!

Иван “2007”
```

Занятие 5 - Ввод и вывод

На последнем занятии, **Ввод и вывод**, вы научитесь:

- Описывать методы консольного вывода в C#.

- Описывать операции консольного ввода в C#.
- Объяснять спецификаторы форматирования в C#.
- Объяснять спецификаторы форматирования даты и времени.

Консольные операции

- Консольные операции - это задачи, выполняемые в интерфейсе командной строки с использованием исполняемых команд.
- Консольные операции используются в программном обеспечении, потому что они легко контролируются операционной системой.
- Консольные операции легко контролируются операционной системой потому что они зависят от устройств ввода и вывода компьютерной системы.
- Консольные приложения выполняются в командной строке.
- Все консольные приложения состоят из трех потоков, представляющих собой последовательности байт.
- Эти потоки связаны с устройствами ввода и вывода компьютерной системы и обрабатывают операции ввода и вывода.

Потоки:

- **Стандартный in**

Стандартный поток in берет ввод и передает его в консольное приложение для обработки.

- **Стандартный out**

Стандартный поток out показывает вывод на мониторе.

- **Стандартный err**

Стандартный поток err показывает сообщения об ошибках на мониторе.

Методы вывода 1-2

- В C# все консольные операции содержатся в классе Console пространства имен System.
- Для вывода данных в консоль вам необходим стандартный поток вывода.
- Стандартный поток вывода предоставляется методами вывода класса Console. Следующие методы вывода пишут в стандартный поток вывода:

- Console.Write()
- Console.WriteLine()

Console.Write()

- Записывает любой тип данных.

Синтаксис

```
Console.Write("data" + имена переменных);
```

где,

data: Заданная строка или символы escape-последовательности, заключенные в двойные кавычки.

Методы вывода 2-2

Console.WriteLine()

- Выводит любой тип данных и символ конца строки в стандартный поток вывода.
- Это означает, что после данных добавляется новая строка.

Синтаксис

```
Console.WriteLine("data" + имена переменных);
```

Пример

```
Console.WriteLine("C# - это мощный язык программирования");
Console.WriteLine("C# мощный");
Console.WriteLine("язык программирования");
Console.Write("C# - это мощный");
Console.WriteLine("язык программирования");
```

В коде показана разница между двумя методами.

Вывод

```
C# - это мощный язык программирования
C# мощный
язык программирования
C# - это мощный язык программирования
```

Указатели места заполнения

- Методы WriteLine() и Write() принимают список параметров для форматирования текста перед выводом.
- Первый параметр - это строка, содержащая маркеры в фигурных скобках, указывающие позицию, которая будет замещена значениями переменной.

- Каждый маркер указывает индекс (начиная с нуля) на основании номеров переменных в списке.
- Например, для указания позиции первого параметра вы напишите {0}, второго {1} и так далее. Числа в фигурных скобках называются указателями места заполнения.

Пример

```
int n, r;
number = 5;
result = 100 * n;
Console.WriteLine("Результат равен {0}, после умножения 100 на {1}", r, n);
r = 150 / n;
Console.WriteLine("Результат равен {0}, после деления 150 на {1}", r, n);
```

Вывод

```
Результат равен 500, после умножения 100 на 5
Результат равен 30, после деления 150 на 5
```

Методы ввода

- В C# для чтения данных вам необходим стандартный поток ввода.
- Этот поток предоставляется методами ввода класса Console.
- Методы ввода, которые позволяют программному обеспечению получить информацию из стандартного устройства ввода:
  - **Console.Read()** - Читает один символ.
  - **Console.ReadLine()** - Читает строку.

Пример

```
string name;
Console.Write("Введите ваше имя: ");
name = Console.ReadLine();
Console.WriteLine("Вас зовут {0}",name);
```

Вывод

```
Введите ваше имя: Иванов Петр
Вас зовут Иванов Петр
```

Методы преобразования

- Метод ReadLine() может также использоваться для ввода целых значений.
- Данные принимаются как строка и затем преобразуются в тип данных int.

- C# предоставляет класс Convert в пространстве имен System для преобразования одних базовых типов данных в другие.

Пример

```
string Name;
int age;
double sal;
Console.Write("Введите ваше имя: ");
Name = Console.ReadLine();
Console.Write("Введите ваш возраст: ");
age = Convert.ToInt32(Console.ReadLine());
Console.Write("Введите вашу зарплату: ");
sal = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Имя: {0}, Возраст: {1}, Зарплата: {2} ", Name,
age, sal);
```

Вывод

```
Введите ваше имя: Петр
Введите ваш возраст: 34
Введите вашу зарплату: 3450.50
Имя: Петр, Возраст: 34, Зарплата: 3450.50
```

Определение спецификаторов числового форматирования

- Спецификаторы форматирования - это специальные символы, которые используются для вывода значений переменных в специальном формате.
- Например, вы можете вывести восьмеричное значение как десятеричное, используя спецификаторы форматирования.
- В C# вы можете преобразовывать числовые значения в различные форматы.
- Например, вы можете выводить большие числа в экспоненциальном формате.
- Для преобразования числовых значений с использованием спецификаторов числового форматирования, вы должны заключить спецификаторы в фигурные скобки.
- Фигурные скобки должны быть заключены в двойные кавычки.
- Это используется в методах вывода класса `Console`.

Синтаксис

```
Console.WriteLine("{спецификатор форматирования}", имя переменной);
```

Некоторые спецификаторы числового форматирования

- Числовые спецификаторы форматирования работают только с числовыми данными.
- Числовые спецификаторы форматирования могут заканчиваться цифрой.
- Эта цифра указывает количество нулей, вставляемых после decimal location.
- Например, если вы используете спецификатор вида C3, три нуля будут вставлены после decimal location для данного числа.

Спецификатор форматирования	**Название**	**Описание**
C или c	Денежный	Число преобразуется в строку, которая представляет денежную сумму.
D или d	Десятичный	Число преобразуется в строку из десятичных цифр (0-9) с префиксом в виде минуса, если число отрицательное. Спецификатор точности указывает минимальное количество цифр, необходимых в результирующей строке. Этот формат поддерживается только для основных типов.
E или e	Научный (Экспоненциальный)	Числа преобразовываются в форму “-d.ddd...E+ddd” или “-d.ddd...e+ddd”, где каждая ‘d’ обозначает цифру (0-9).

Пример

```
int num = 456;
Console.WriteLine(“{0:C}”, num);
Console.WriteLine(“{0:D}”, num);
Console.WriteLine(“{0:E}”, num);
```

Вывод

```
$456.00
456
4.560000E+002
```

Еще числовые спецификаторы форматирования

Спецификатор форматирования	Название	Описание
F или f	С фиксированной точкой	Числа преобразовываются в форму “-ddd.ddd...” где каждая ‘d’ обозначает цифру (0-9). Если число отрицательное, строка начинается с символа минус.
N или n	Число	Числа преобразовываются в форму “-d,ddd,ddd.ddd...” где каждая ‘d’ обозначает цифру (0-9). Если число отрицательное, строка начинается с символа минус.
X или x	Шестнадцатеричное	Число преобразуется в строку шестнадцатеричных цифр. Используйте “X” для “ABCDEF” и “x” для “abcdef”.

Пример

```
int num = 456;
Console.WriteLine(“{0:F}”, num);
Console.WriteLine(“{0:N}”, num);
Console.WriteLine(“{0:X}”, num);
```

Вывод

```
456.00
456.00
1C8
```

Стандартные спецификаторы форматирования даты и времени

- Спецификаторы форматирования даты и времени - это специальные символы, позволяющие вам выводить значения даты и времени в различных форматах.

- Например, вы можете вывести дату в виде mm-dd-yyyy и время hh:mm. Если вы выводите Гринвичское время (GMT), то вы можете выводить его с использованием аббревиатуры GMT, используя спецификаторы форматирования даты и времени.
- Спецификаторы форматирования даты и времени позволяют вам выводить дату и время в 12-ти и 24-х часовых форматах.

Синтаксис

```
Console.WriteLine("{спецификатор форматирования}", объект класса DateTime);
```

Некоторые стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
d	Короткая дата	Выводит дату в соответствии с коротким шаблоном. Формат по умолчанию “mm/dd/yyyy”.
D	Длинная дата	Выводит дату в соответствии с длинным шаблоном. Формат по умолчанию “dddd*, MMMM*, dd, yyyy”.
f	Полностью дата/время (короткое время)	Выводит дату в длинном формате, дата и время разделены пробелом. Формат по умолчанию “dddd*, MMMM* dd, yyyy HH*:mm*”.
F	Полностью дата/время (длинное время)	Выводит дату в длинном формате и время в длинном формате (разделенные пробелом). Формат по умолчанию “dddd*, MMMM* dd, yyyy HH*: mm*: ss*”.
g	Обычное дата/время (короткое время)	Выводит дату в коротком формате и время коротком формате (разделенные пробелом). Формат по умолчанию “MM/dd/yyyy HH*:mm*”.

Некоторые стандартные спецификаторы форматирования даты и времени 2-2

Пример

```
DateTime dt = DateTime.Now;
Console.WriteLine("Короткая дата(d): {0:d}", dt);
Console.WriteLine("Длинная дата(D): {0:D}", dt);
Console.WriteLine("Длинная дата и время без секунд(f): {0:f}", dt);
Console.WriteLine("Короткая дата и время с секундами(F): {0:F}", dt);
Console.WriteLine("Короткая дата и время без секунд(g): {0:g}", dt);
```

Вывод

```
Короткая дата(d): 23/04/2007
Длинная дата(D): Monday, April 23, 2007
Длинная дата и время без секунд(f): Monday, April 23, 2007 12:58 PM
Длинная дата и время с секундами(F): Monday, April 23, 2007 12:58:43
PM
Короткая дата и время без секунд(g): 23/04/2007 12:58 PM
```

Еще стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
G	Обычное дата/время (длинное время)	Выводит дату в коротком формате и время в длинном формате (разделенные пробелом). Формат по умолчанию “MM/dd/yyyy HH*:mm*:ss*”.
m или M	День месяца	Выводит только месяц и день. Формат по умолчанию “MMMM* dd”.
T	Короткое время	Выводит время в соответствии с коротким шаблоном. Формат по умолчанию “HH*: mm*”.
T	Длинное время	Выводит время в соответствии с длинным шаблоном. Формат по умолчанию “HH*:mm*:ss*”.
y или Y	Шаблон месяцев года	Выводит только месяц и год. Формат по умолчанию “YYYY MMMM*”.

Еще стандартные спецификаторы форматирования даты и времени 2-2

Пример

```
DateTime dt = DateTime.Now;
Console.WriteLine("Короткая дата и время с секундами(G): {0:G}", dt);
// Возвращает месяц и день - M также может быть использовано
Console.WriteLine("Месяц и день(m): {0:m}", dt);
Console.WriteLine("Короткое время(t): {0:t}", dt);
Console.WriteLine("Короткое время с секундами(T):
{0:T}", dt);
// Возвращает год и месяц - Y также может быть использовано
Console.WriteLine("Год и месяц (y): {0:y}", dt);
```

Вывод

```
Короткая дата и время с секундами(G): 23/04/2007 12:58:43 PM
Месяц и день(m): April 23
Короткое время(t): 12:58 PM
Короткое время с секундами(T): 12:58:43 PM
Год и месяц(y): April, 2007
```

Заключение 1-3

Переменные и типы данных в C#

- Переменные - это именованные области компьютерной памяти и хранилища данных.
- Вы можете присвоить имя и тип данных переменной.
- Тип данных указывает вид данных, хранящихся в переменной.
- Тип данных может быть любым типом-значением или ссылочным типом.

Комментарии и XML-документация

- Комментарии используются для предоставления детальных пояснений о различных строках кода.
- Вы можете вставить комментарий, добавив двойной обратный слэш (//) перед пояснением к коду.
- XML-комментарии начинаются с тройного обратного слэша (///) и заключаются в XML-тэги.

- Вы можете создать XML-документ, который будет содержать все XML-комментарии.

Заключение 2-3

Константы и литералы

- Константы - это статические значения, которые вы не можете изменять при выполнении программы.
- Литералы - это фиксированные значения, которые вы должны вводить с клавиатуры.
- Вы можете определять константы и литералы для любого типа данных.

Ключевые слова и Escape последовательности

- Ключевые слова - это специальные слова предопределенные в C#.
- Вы не можете использовать их как имена переменных, методов или классов.
- Escape-последовательности - это специальные символы, которым предшествует обратный слэш.
- Символы escape-последовательностей позволяют выводить непечатные символы.

Заключение 3-3

Ввод и вывод

- Console.Read() и Console.ReadLine() - методы ввода, которые всегда возвращают строковое значение.
- Вы можете преобразовать строковое значение в другой тип данных, используя методы преобразования класса Convert.
- Console.Write() и Console.WriteLine() - методы вывода.
- Спецификаторы форматирования позволяют вам настроить вывод в окно консоли.

Текст презентации к лекции №3

Операции и операторы

Обзор модуля

В этом модуле вы изучите:

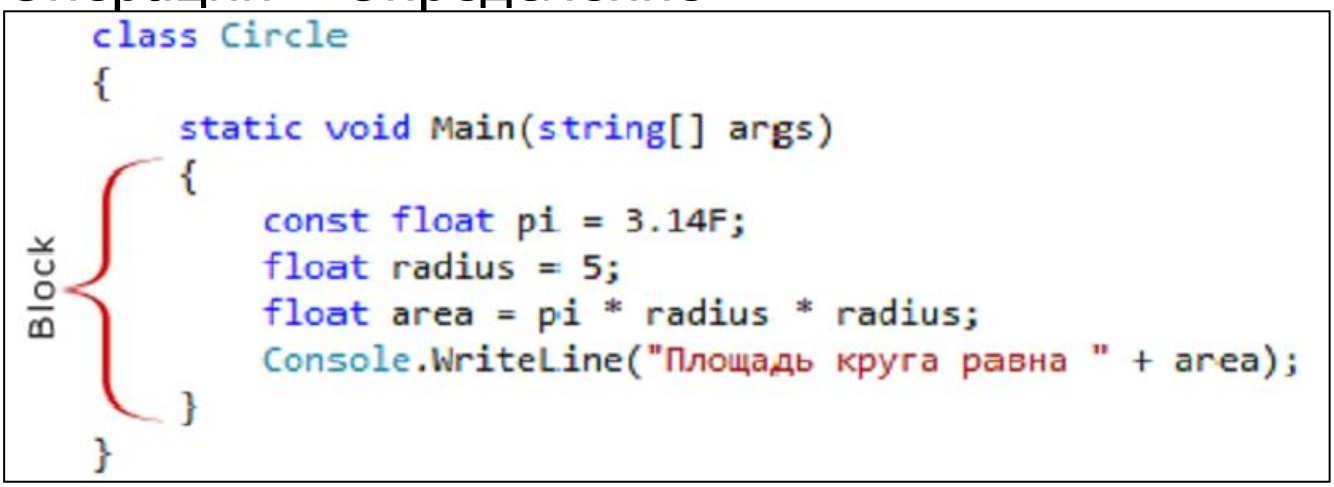
- Операции и выражения
- Типы операторов
- Преобразование данных в C#

Занятие 1 -Операции и выражения

На первом занятии, **Операции и выражения**, вы научитесь:

- Описывать операции и их цели.
- Перечислять категории операций.
- Описывать и использовать выражения в C#.

Операции – Определение




```
class Circle
{
    static void Main(string[] args)
    {
        const float pi = 3.14F;
        float radius = 5;
        float area = pi * radius * radius;
        Console.WriteLine("Площадь круга равна " + area);
    }
}
```

- Операции представляют собой логическую группу из переменных, операторов и ключевых слов C#, выполняющих специальную задачу.
- В C# операции заканчиваются точкой с запятой (;).
- Программы строятся из множества операций, группируемых в блоки.

Операции - Использование

- Операции используются для определения ввода, обработки и вывода в программе.
- Операции могут включать:
  - Типы данных
  - Переменные
  - Операторы
  - Константы
  - Литералы
  - Ключевые слова
  - Символы escape-последовательностей
  - Операции помогают вам построить логический поток в программе.
  - С помощью операций вы можете:
  - Инициализировать переменные и объекты
  - Получать ввод
  - Вызывать методы классов

- Выполнять вычисления
- Отображать вывод

Пример

```
double area = 3.1452 * radius * radius;
```

- Операция вычисляет площадь круга и сохраняет значение в переменную area.

```
{
    int side = 10;
    int height = 5;
    double area = 0.5 * side * height;
    Console.WriteLine("Площадь: " , area);
}
```

- Вышеуказанный пример показывает блок кода, заключенный в фигурные скобки
- Операции выполняются последовательно, начиная с первой верхней.

- Следующий пример демонстрирует вложенность блоков в C#.

```
{
    int side = 5;
    int height = 10;
    double area;
    {
    area = 0.5 * side * height;
    }
    Console.WriteLine(area);
}
```

- Первые три операции (начиная сверху) будут выполнены последовательно.
- Затем выполняется вычисляющий площадь код в фигурных скобках.
- Выполнение завершается, когда последняя операция в блоке кода выводит площадь.

Типы операций

- Операции C# похожи на операции в C и C++.

- Операции C# классифицируются по нескольким категориям:
  - Операции выбора
  - Операции повторения
  - Операции перехода
  - Операции обработки исключений
  - Ограниченные и не ограниченные операции
  - Фиксированные операции
  - Блокирующиеся операции

  - Операции обработки исключений - управляют неожиданными ситуациями, которые затрудняют нормальное выполнение программы.
  -
  - Ограниченные и неограниченные операции - управляют арифметическим переполнением.
  -
  - Фиксированные операции - требуются для указания сборщику мусора не удалять объект во время выполнения.
  -
  - Блокирующиеся операции - помогают закрыть критические блоки кода.
  -
  - Операции выбора - это операции принятия решения, которые проверяют, является ли заданное условие истинным или ложным.
  -
  - Операции повторения - помогают вам повторно выполнить блок кода.
  -

- Операции перехода - помогают вам привести поток из одного блока программы в другой блок.

Выражения - Определение

- Выражения используются для манипулирования данными.
- Выражения в C# заканчиваются точкой с запятой (;).
- Выражения:
  - Вычисляют значения
  - Получают результаты вычислений
  - Служат частью других выражений или операций

Пример

- В первых двух строчках кода результат операций сохраняется в переменные SimpleInterest и eval.
- Последняя операция увеличивает значение переменной num.

Различия между операциями и выражениями

Операции	**Выражения**
Не обязаны возвращать значения. Например: int oddNum = 5; Эта операция сохраняет значение 5 в переменную oddNum.	Всегда возвращает значение. Например: 10000*(75/100) Это выражение вернет значение 7500.
Задает переменную для хранения результатов вычислений. Например: int evenNum=(10 *100)/5; Эта операция сохраняет результат вычислений в переменную evenNum.	Не задает переменную для хранения результатов вычислений. Например: (8.5+9.4) * (1034.56/6) Это выражение вычисляет значение без сохранения его в какую-либо переменную. Для сохранения значения, вы должны присвоить выражение переменной. Это

	требует использования операции. Выражение само по себе не сохраняет значений.
Операции выполняются компилятором.	Выражения - это часть операций, они вычисляются компилятором.

Занятие 2 - Типы операторов

На втором занятии, **Типы операторов**, вы научитесь:

- Описывать операторы и их цели.
- Определять и объяснять использование арифметических операторов.
- Определять и объяснять использование операторов отношений, логических операторов и операторов сравнения.
- Определять и объяснять использование операторов инкремента и декремента.
- Определять и объяснять использование операторов присваивания.
- Объяснять порядок выполнения операторов.

Что такое операторы?

- В операциях действия выполняются над одним или несколькими значениями, сохраненными в переменных, для их модификации или генерации нового значения.
- Операции выполняются при помощи как минимум одного символа и значения.
- Этот символ называется оператором и определяет тип действия, выполняемого над значением.
- Значения, над которыми выполняется операция, называются операндами.

Типы операторов

- В C# предопределен набор операций, используемых для выполнения различных типов операций.
- Операторы в C# разделяются на шесть категорий по типу действий, выполняемых над значениями:
  - Арифметические операторы
  - Операторы отношения
  - Логические операторы
  - Условные операторы
  - Операторы декремента и инкремента
  - Операторы присваивания

Арифметические операторы - типы

- Арифметические операторы являются двоичными, поскольку они работают с двумя операндами (оператор помещается между операндами).

Операторы	**Описание**	**Пример**

+ (Сложение)	Выполняет сложение. Если операнды - строки, то работает как оператор конкатенации строк и добавляет одну строку в конец другой.	40 + 20
- (Вычитание)	Выполняет вычитание. Если большее значение вычитается из меньшего, в результате получится отрицательное значение.	100 - 47
* (Умножение)	Выполняет умножение.	67 * 46

Операторы	**Описание**	**Пример**
/ (Деление)	Выполняет деление. Оператор делит первый операнд на второй и возвращает частное как результат.	12000 / 10
% (Остаток от деления)	Выполняет операцию вычисления остатка от деления. Оператор делит два операнда и возвращает остаток от деления как результат.	100 % 33

Пример

```
int valueOne = 10;
int valueTwo = 2;
int add = valueOne + valueTwo;
int sub = valueOne - valueTwo;
int mult = ValueOne * valueTwo;
int div = valueOne / valueTwo;
int modu = valueOne % valueTwo;
Console.WriteLine("Сложение " + add );
Console.WriteLine("Вычитание " + sub);
Console.WriteLine("Умножение " + mult);
Console.WriteLine("Деление " + div);
Console.WriteLine("Остаток " + modu);
```

Вывод

```
Сложение 12
Вычитание 8
Умножение 20
Деление 5
Остаток 0
```

Операторы отношения

- Операторы отношения сравнивают два операнда и возвращают логическое значение

true или false.

Операторы отношения	**Описание**	**Пример**
==	Проверяет равенство операндов.	85 == 95
!=	Проверяет, не равны ли операнды.	35 != 40
>	Проверяет, больше ли первое значение второго.	50 > 30
<	Проверяет, меньше ли первое значение второго.	20 < 30
Операторы отношения	Описание	Пример
>=	Проверяет, больше или равно первое значение второму.	100 >= 30
<=	Проверяет, меньше или равно первое значение второму.	75 <= 80

Пример

```
int leftVal = 50;

int rightVal = 100;

Console.WriteLine("Равно: " + (leftVal ==
rightVal));

Console.WriteLine("Не равно: " + (leftVal !=
rightVal));

Console.WriteLine("Больше: " + (leftVal >
rightVal));

Console.WriteLine("Меньше: " + (leftVal <
rightVal));

Console.WriteLine("Больше или равно: " +
(leftVal >= rightVal));

Console.WriteLine("Меньше или равно: " +
(leftVal <= rightVal));
```

Вывод

```
Равно: False

Не равно: True

Больше: False

Меньше: True

Больше или равно: False

Меньше или равно: True
```

Логические операторы 1-4

- Логические операторы - это двоичные операторы, которые выполняют логические операции над двумя операндами и возвращают логическое булево значение.

- C# поддерживает два типа логических операторов:
  - Булевы
  - Битовые

Булевы логические операторы

Булевы логические операторы

- выполняют булевы логические операции над обоими аргументами.
- возвращают булево значение в зависимости от использованного логического оператора.

- В таблице перечислены булевы логические операторы и их описания.

Логические операторы	**Описание**	**Пример**
& (Булево и)	Возвращает true, если оба выражения true.	(percent >= 75) & (percent <= 100)
| (Булево включающее или)	Возвращает true, если хотя бы одно из выражений равно true.	(choice == 'Y') | (choice == 'y')
^ (Булево исключающее или)	Возвращает true, если только одно из выражений равно true. Если оба выражения равны true, оператор возвратит false.	(choice == 'Q') ^ (choice == 'q')

Логические операторы 2-4

Пример - Оператор включающее или

```
if ((quantity == 2000) ^ (price == 10.5))

{

    Console.WriteLine ("Вы должны пойти на
компромисс между количеством и ценой");
```

Пример - Оператор и

```
if ((quantity == 2000) & (price == 10.5))
{
    Console.WriteLine ("Товары по правильной
```

Пример - Оператор исключающее или

```
if ((quantity > 2000) | (price < 10.5))
{
    Console.WriteLine ("Вы можете купить больше
товаров по      более низкой цене");
```

Логические операторы 3-4

Битовые логические операторы

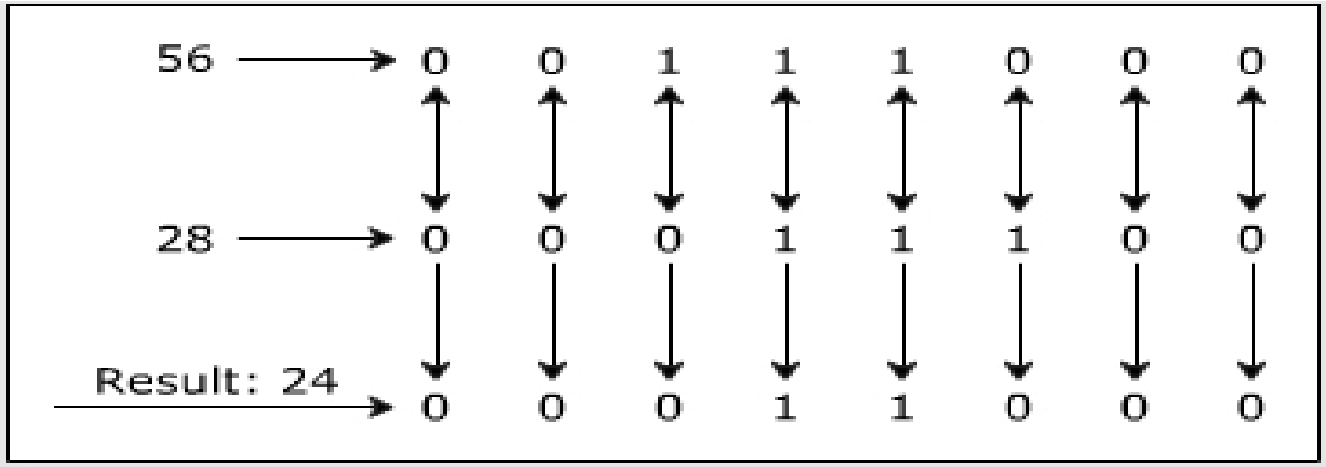
- Битовые логические операторы выполняют логические операции над отдельными битами операндов.

Логические операторы	Описание	Пример
& (Битовое и)	Сравнивает два бита и возвращает 1, если оба бита 1, иначе возвращает 0.	00111000 & 00011100
| (Битовое включающее или)	Сравнивает два бита и возвращает 1, если хотя бы один равен 1.	00010101 | 00011110
^ (Битовое исключающее или)	Сравнивает два бита и возвращает 1, если только один равен 1.	00001011 ^ 00011110

Логические операторы 4-4

Пример - Битовый оператор и

```
if ((quantity > 2000) | (price < 10.5))
{
    Console.WriteLine ("Вы можете купить больше товаров по более низкой цене");
```



Пример - Битовый оператор включающее или

```
result = 56 | 28;
Console.WriteLine(result);
```

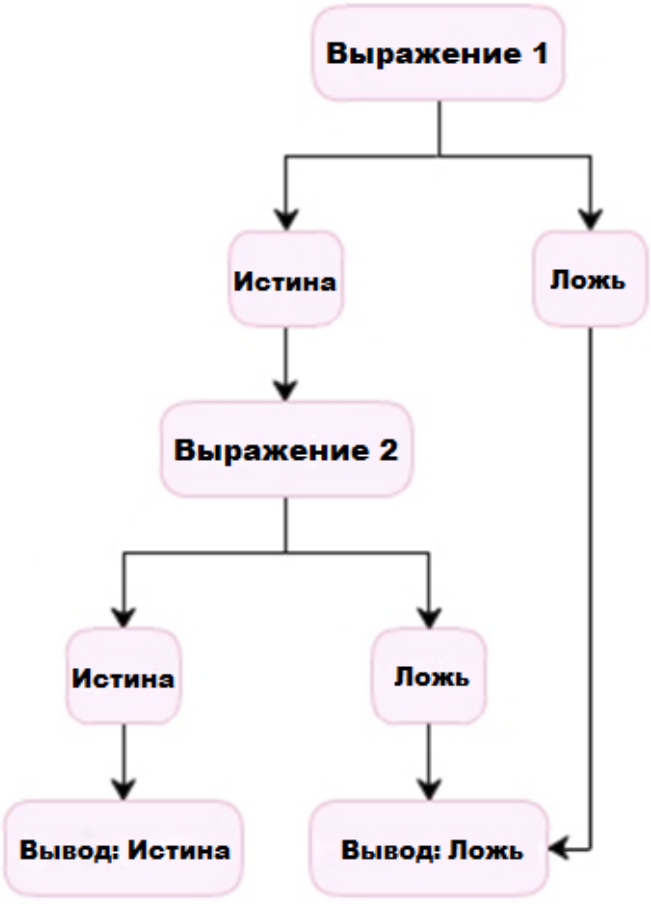
Пример - Битовый оператор исключающее или

```
result = 56 ^ 28;
Console.WriteLine(result);
```

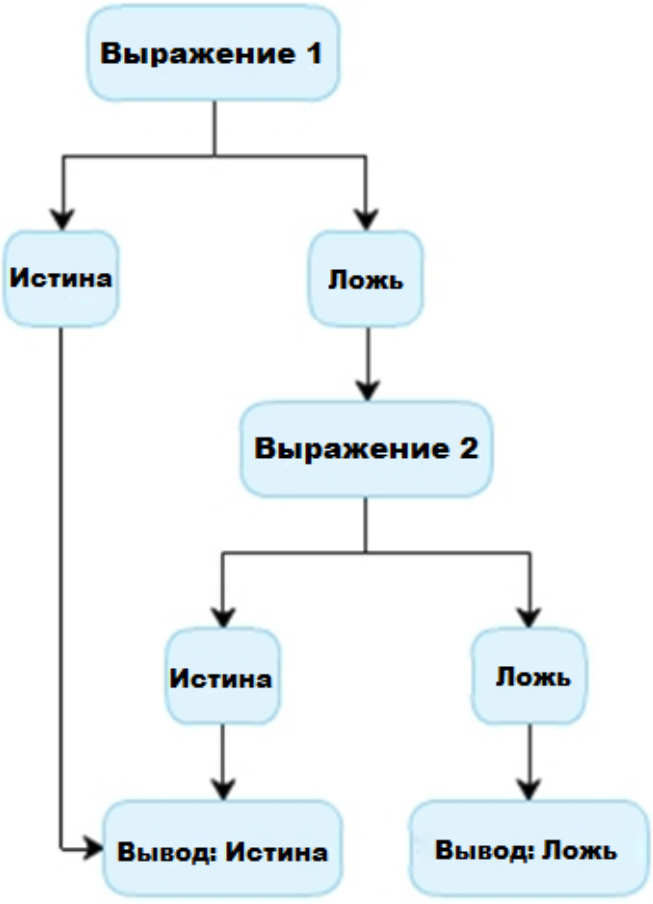
Условные операторы

- Существует два типа условных операторов: условное «и» (&&) и условное «или» (||).

- Условные операторы похожи на булевы логические операторы, но имеют следующие отличия:
  - Условный оператор «и» вычисляет значение второго выражения, только если первое выражения возвратило true.
  - Условный оператор «или» вычисляет значение второго выражения, только если первое выражения возвратило false.



Условно И



Условно ИЛИ

Пример

```
int num = 0;
if (num >= 1 && num <= 10)
{
    Console.WriteLine("Число входит в промежуток от 1 до 10");
}
else
{
    Console.WriteLine("Число не входит в промежуток от 1 до 10");
}
```

Вывод

```
Число не входит в промежуток от 1 до 10
```

Пример

```
int num = -5;

if (num < 0 || num > 10)

{

    Console.WriteLine("Число не входит в
промежуток от 1     до 10");

}

else

{

    Console.WriteLine("Число входит в промежуток
от 1 до     10");

}
```

Вывод

```
Число не входит в промежуток от 1 до 10
```

Операторы декремента и инкремента

- Два наиболее часто выполняемых вычисления в программировании - это инкрементация и декрементация значения переменной на 1.
- В C# оператор инкремента (++) используется для увеличения значения на 1, тогда как оператор декремента (--) используется для уменьшения значения на 1.

- В таблице приведен пример использования инкремента и декремента, при условии значения переменной `valueOne` 5.

Выражение	**Тип**	**Результат**
valueTwo = ++ValueOne;	пред-инкремент	valueTwo = 6
valueTwo = valueOne++;	пост-инкремент	valueTwo = 5
valueTwo = --valueOne;	пред-декремент	valueTwo = 4
valueTwo = valueOne--;	пост-декремент	valueTwo = 5

Операторы присваивания - типы

- Операторы присваивания используются для присвоения значения правого операнда левому с использованием оператора эквивалентности (=).

- В C# операторы присваивания делятся на две категории:
  - Простые операторы присваивания
  - Составные операторы присваивания

  - Простые операторы присваивания: Простой оператор присваивания это "=", он используется для присвоения значения или результата выражения переменной.
  -
  - Составные операторы присваивания: Они получаются из комбинации простого оператора присваивания и арифметического оператора.
  -
  - В таблице приведен пример использования операторов присваивания, при условии значения переменной valueOne10.

Выражение	**Описание**	**Результат**
valueOne += 5;	valueOne = valueOne + 5	valueOne = 15
valueOne -= 5;	valueOne = valueOne - 5	valueOne = 5
valueOne *= 5;	valueOne = valueOne * 5	valueOne = 50
valueOne %= 5;	valueOne = valueOne % 5	valueOne = 0
valueOne /= 5;	valueOne = valueOne / 5	valueOne = 2

Пример

```
int valueOne = 5;

int valueTwo = 10;

Console.WriteLine("Значение1 =" + valueOne);

valueOne += 4;

Console.WriteLine("Значение1 += 4= " +
valueOne);

valueOne -= 8;

Console.WriteLine("Значение1 -= 8= " +
valueOne);

valueOne *= 7;

Console.WriteLine("Значение1 *= 7= " +
valueOne);

valueOne /= 2;

Console.WriteLine("Значение1 /= 2= " +
valueOne);

Console.WriteLine("Значение1 == Значение2: {0}",
(valueOne == valueTwo));
```

```
Значение1 =5

Значение1 += 4= 9

Значение1 -= 8= 1

Значение1 *= 7= 7

Значение1 /= 2= 3

Значение1 == Значение2: False
```

Приоритет и ассоциативность

- Операторы в C# имеют соответствующие уровни приоритета.
- Компилятор C# выполняет операторы в последовательности, определенной уровнем приоритета операторов.
- В таблице перечислены приоритеты операторов, их описание и ассоциативность.

Приоритет **(1 является максимальным)**	**Оператор**	**Описание**	**Ассоциативность**
1	()	Круглые скобки	Слева направо
2	++ или --	Инкремент или декремент	Справа налево
3	*, /, %	Умножение, деление, остаток от деления	Слева направо
4	+, -	Сложение, вычитание	Слева направо
5	<, <=, >, >=	Меньше, меньше или равно, больше, больше или равно	Слева направо
6	=, !=	Равно, не равно	Слева направо
7	&&	Условное и	Слева направо
8	||	Условное или	Слева направо
9	=, +=, -=, *=, /=, %=	Операторы присваивания	Справа налево

Пример

```
int valueOne = 10;

Console.WriteLine((4 * 5 - 3 ) / 6 + 7 - 8 % 5);

Console.WriteLine((32 < 4) || (8 == 8));

Console.WriteLine(((valueOne *= 6) > (valueOne
+= 5)) &&

((valueOne /= 2) != (valueOne -= 5)));
```

Вывод

```
6

True

False
```

Занятие 3 - Преобразование данных в C#

На последнем занятии, **Преобразование данных в C#**, вы научитесь:

- Описывать виды приведения типов и их преимущества.
- Описывать неявное преобразование типов.
- Описывать явное преобразование типов.
- Объяснять упаковку и распаковку.

Приведение типов и его преимущества - Необходимость

- Отдел оплаты труда требует, чтобы зарплата выводилась целым числом, т.е. числа после запятой при расчетах игнорировались.
- Программист может добиться этого, используя приведение типов в C#.
- Приведение типов позволяет изменять тип данных переменной.

Приведение типов и его преимущества - Определение и преимущества

- C# поддерживает два типа приведения типов: явные и неявные.

- Приведение типов главным образом используется для:
  - Преобразования типа данных в другой тип данных, который принадлежит той же или другой иерархии.
  - Отображения точного числового вывода.
  - Предотвращения потери числовых данных, если результирующее значение выходит за пределы диапазона значений типа данных.

Неявное преобразование для типов данных C# - Определение

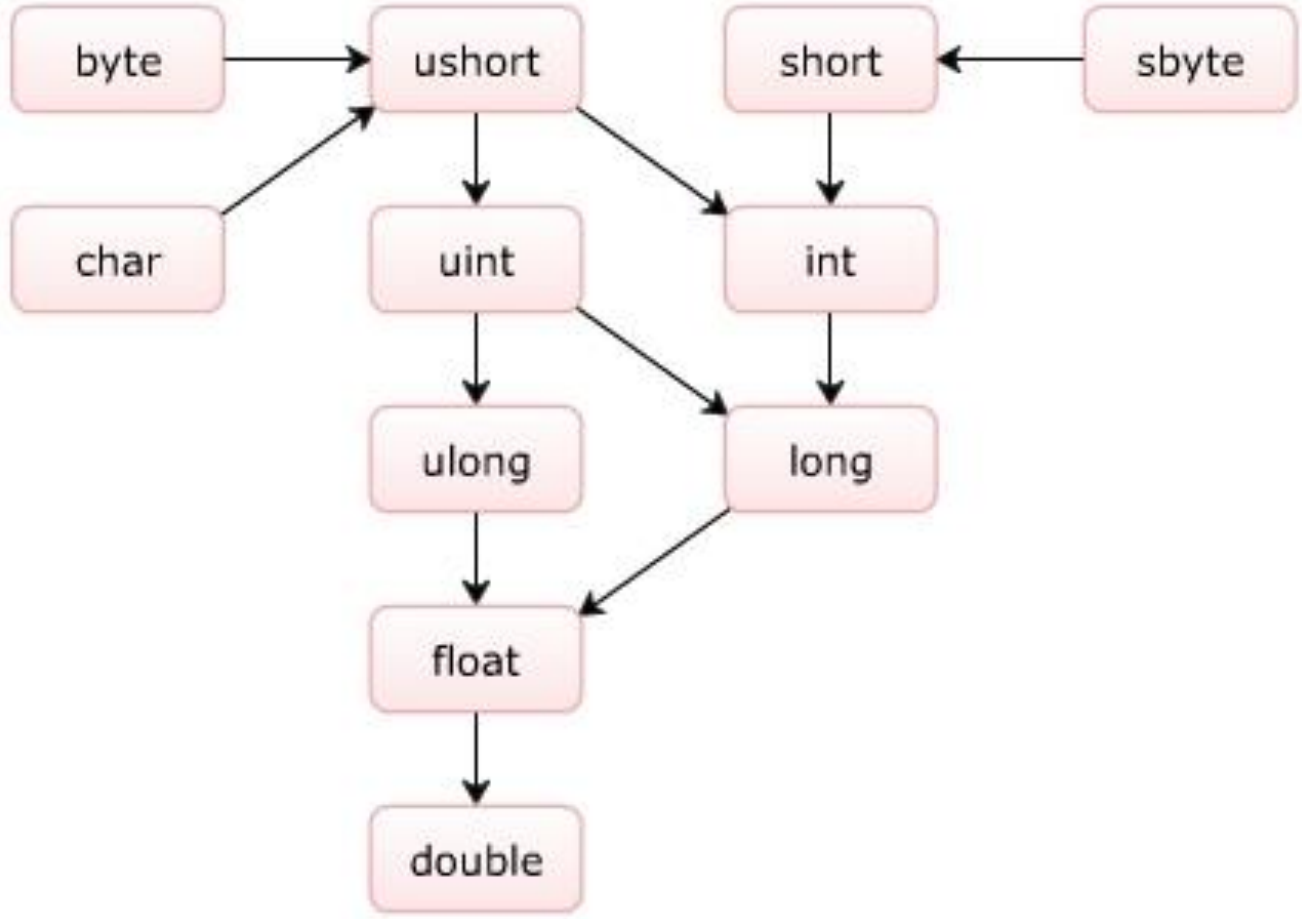
- Это автоматическое преобразование типов данных.
- Оно производится, только если исходный и результирующий тип данных находятся в одной иерархии.

Неявное преобразование для типов данных C# - Правила

- Производится компилятором автоматически.

- Компилятор C# автоматически конвертирует типы данных меньшей точности в типы данных большей точности.

- График показывает различные типы данных и типы данных с большей точностью, в которые они могут быть конвертированы.



Явное приведение типов - Определение

- Явное приведение типов позволяет менять тип данных с более точного на менее точный.
- Используя явное приведение типов, вы можете вручную преобразовать значение типа `float` в тип `int`.

Синтаксис

```
<target data type> <variable name> = (target data type)<source data type>;
```

где, target data type: **результирующий тип данных.**

variable name: **имя переменной результирующего типа данных.**

target data type: **результирующий тип данных в круглых скобках.**

source data type: **тип данных из которого производится преобразование.**

Пример

```
double side = 10.5;

int area;

area = (int)(side * side);

Console.WriteLine("Площадь квадрата = {0}", area);
```

Вывод

```
Площадь квадрата = 110
```

Явное приведение типов - Реализация

- В C# есть два пути реализации явного преобразования типов с использованием встроенных методов:
  - Класс **`System.Convert`**
  - **Метод `ToString()`**

- **Класс `System.Convert`** предоставляет удобные методы для преобразования любых встроенных типов данных в другие встроенные типы данных.

- Метод **`ToString()`** принадлежащий классу `Object` преобразует любой тип данных в строку `string`.

Пример

```
float flotNum = 500.25F;

string stNum = flotNum.ToString();

Console.WriteLine(stNum);
```

Вывод

```
500.25
```

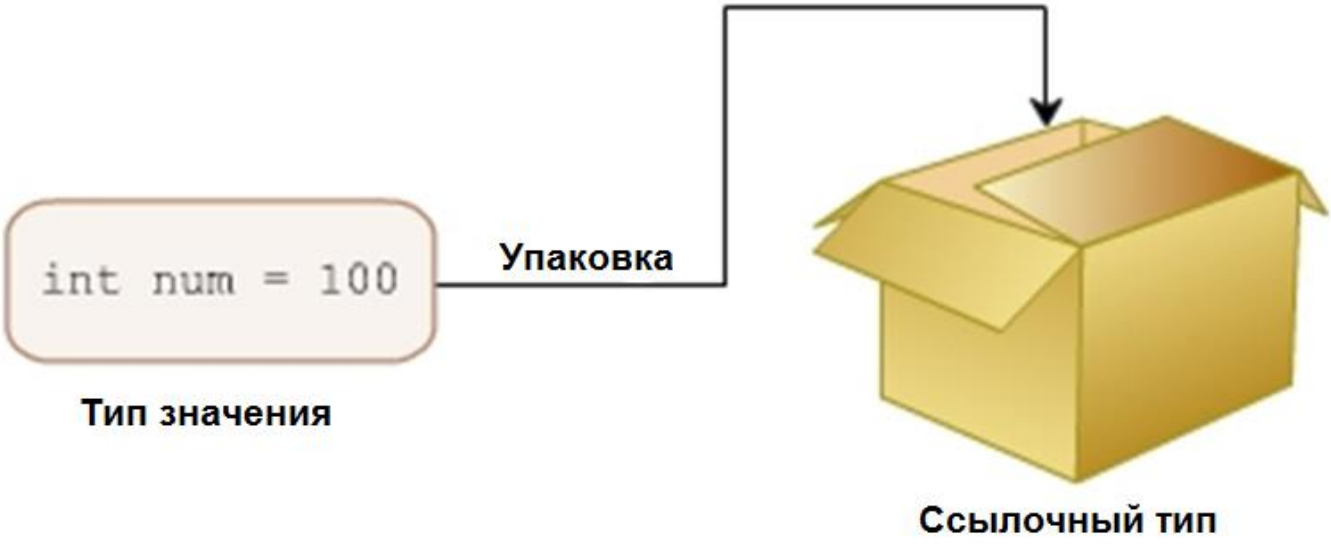
Упаковка

- Это процесс преобразования типа значения, например, integers, в ссылочный тип, например, objects.

- Это преобразование используется для уменьшения издержек системы при выполнении.

- Для реализации упаковки вы должны присвоить тип-значение объекту.

- Упаковка выполняется неявно, когда тип значений предоставляется вместо отсутствующего ссылочного типа.



Синтаксис

```
object <instance of the object class> = <variable of value type>;
```

где,

`object`: базовый класс для всех типов-значений.

`instance of the object class`: имя ссылки на класс `Object`.

`variable of value type`: идентификатор типа данных

- В следующем примере показано использование явной упаковки.

```
float radius = 4.5F;

double circumference;

circumference = 2 * 3.14 * radius;

object boxed = (object)circumference;

Console.WriteLine("Периметр круга= {0}", circumference);
```

Пример – неявная упаковка

```
int radius = 10;

double area;

area = 3.14 * radius * radius;

object boxed = area;

Console.WriteLine("Площадь круга = {0}",boxed);
```

Вывод

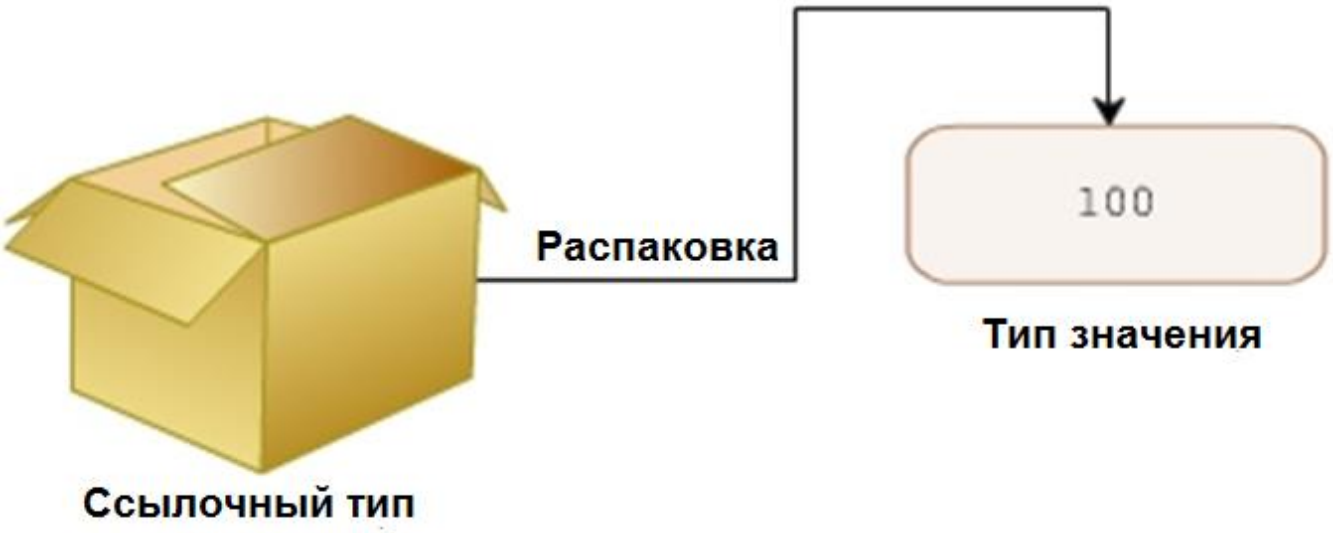
```
Area of the circle = 314
```

Вывод

```
Периметр круга = 28.26
```

Распаковки

- Распаковка представляет собой преобразование ссылочного типа в тип-значение.
- Значение, хранящееся в объекте, распаковывается в тип-значение.



Синтаксис - Распаковка

```
<target value type> <variable name> = (target value type) <object type>;
```

где,

`target value type`: результирующий тип данных.

`variable name`: имя переменной типа данных.

`target value type`: результирующий тип данных в круглых скобках.

`object type`: именованная ссылка на класс `Object`.

Пример

```
int length = 10;

int breadth = 20;

int area;

area = length * breadth;

object boxed = area;

int num = (int)boxed;

Console.WriteLine("Площадь прямоугольника= {0}", num);
```

Заключение 1-2

- **Операции и выражения**
  - Это исполняемые строки кода, которые составляют программу.
- **Типы операторов**
  - Различные типы операторов:
    - **Арифметические операторы**
    - **Операторы отношения**
    - **Логические операторы**
    - **Условные операторы**
    - **Операторы декремента и инкремента**
    - **Операторы присваивания**

Заключение 2-2

- **Преобразование данных в C#**
  - В C# вы можете преобразовывать типы-значения в ссылочные типы, используя метод упаковки.

- Преобразование ссылочного типа в тип-значение называется распаковкой.

Текст презентации к лекции №4-6

Конструкции

Обзор модуля

В этом модуле вы изучите:

- Конструкции выбора
- Циклические конструкции
- Операции перехода в C#

Занятие 1 - Конструкции выбора

На первом занятии, **Конструкции выбора**, вы научитесь:

- Объяснять конструкции выбора и их пользу.
- Объяснять конструкции выбора if и if..else.
- Объяснять конструкции выбора if…else if.
- Формулировать синтаксис и работу вложенных конструкций if.
- Формулировать синтаксис и работу операции switch…case.

Конструкции выбора

- Это поддерживаемые C# программные конструкции, которые позволяют управлять течением программы.

- Они называются конструкциями принятия решений.

- C# поддерживает следующие конструкции принятия решений:
  - Конструкция if else
  - Конструкция if..else if
  - Вложенные конструкции if
  - Конструкция switch…case

Операция “if”

- Позволяет выполнить блок операций после проверки специального логического условия.
- Начинается с ключевого слова if, после которого следует условие.

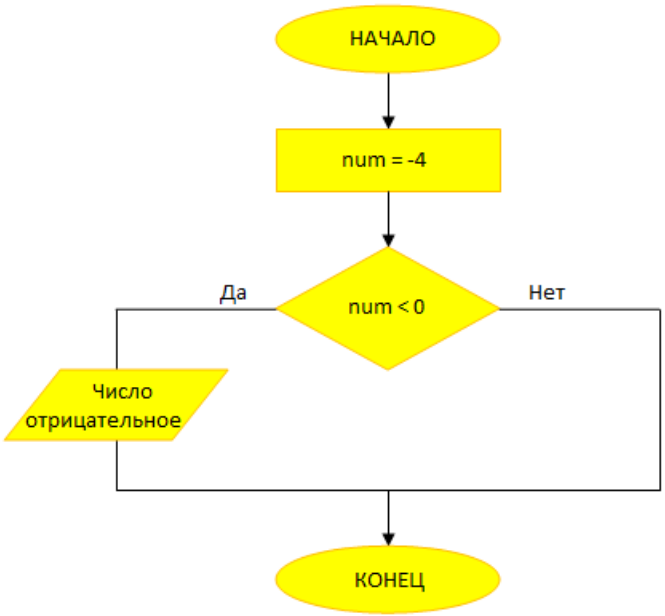
Синтаксис:

```
if (условие)

{

   // одна или более операция;

}
```

где,

условие: булево выражение.

операторы: исполняемые инструкции, выполняемые, когда булево выражение возвращает true.



Пример:

```
public int num = -4;

if (num < 0)

{

   Console.WriteLine(“Число отрицательное”);

}
```

Вывод

Число отрицательное

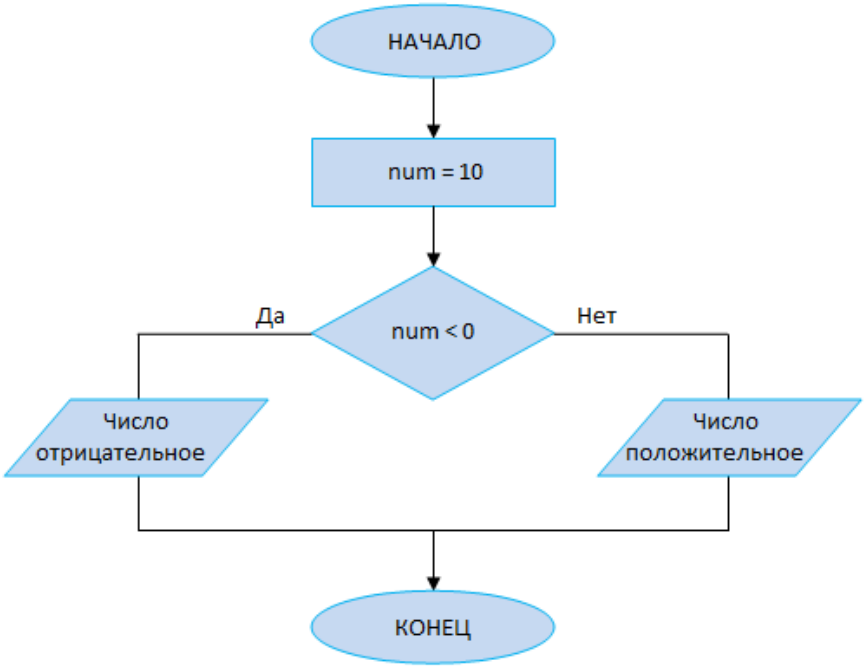
Конструкция “if..else” 1-2

- Выполняет блок операций, только если заданное условие истинно.
- Начинается с if, затем блок кода, затем блок else.
- Блок else начинается с ключевого слова else, за которым следует блок операций.

Синтаксис:

```
if (условие)

{

   // одна или более операция;
```

```
}
else
{
   // одна или более операция;
}
```



Пример:

```
public int num = 10;
if (num < 0)
{
   Console.WriteLine(“Число отрицательное”);
}
else
{
   Console.WriteLine(“Число положительное”);
}
```

Вывод:

Число положительное

Конструкция “if..else if” 2-2

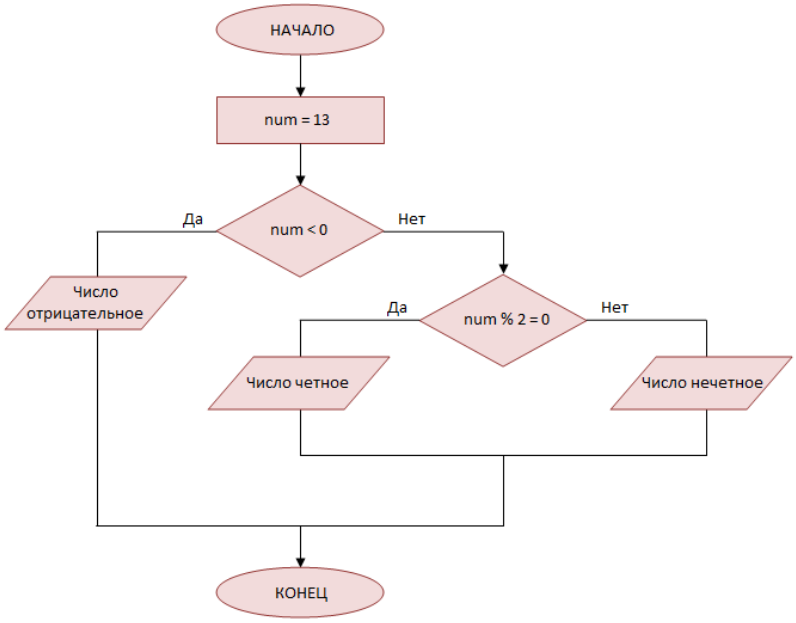
- Позволяет проверить несколько условий и выполнить различные блоки кода для каждого условия.

- Конструкция начинается с операции if, затем следует несколько операций else if, за которыми следует необязательный блок else.

- Условия, заданные в конструкции if..else if, выполняются последовательно.

Синтаксис:

```
if (условие)

{

   // одна или более операция;

}

else if (условие)

{

   // одна или более операция;

}

else

{

   // одна или более операция;

}
```



Пример:

```
public int num = 13;

if (num < 0)

{

   Console.WriteLine(“Число отрицательное”);

}

else if ((num % 2) == 0)

{

   Console.WriteLine(“Число четное”);

}

else

{

   Console.WriteLine(“Число нечетное”);

}
```

Вывод:

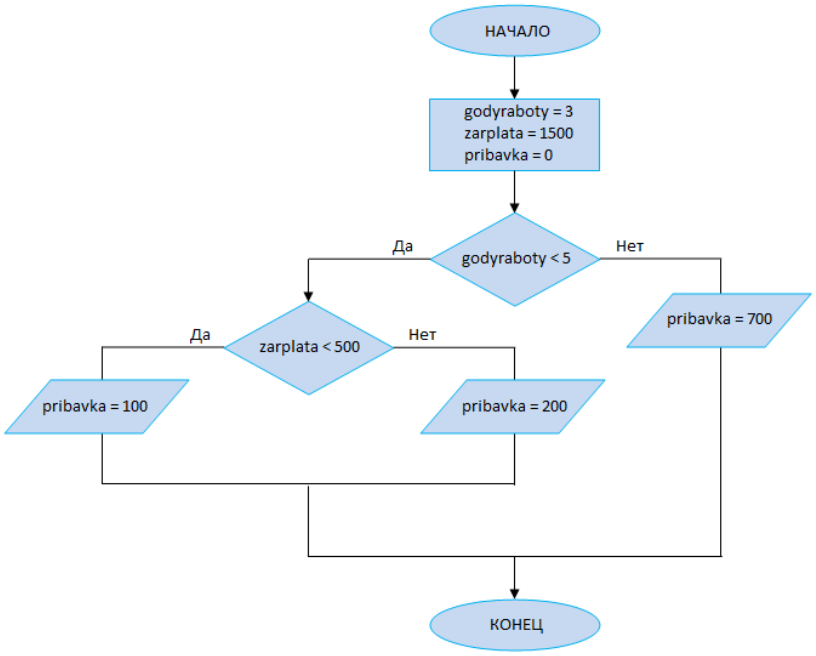
Число нечетное

Вложенные конструкции “if”

- Это конструкции из нескольких операций if.
- Начинается с операции if, которая называется внешней операцией, и содержит несколько операций if, которые называются вложенными операциями.
- Условие внешнего if контролирует выполнение вложенной операции if.

Синтаксис:

```
if (условие)
{
   // одна или более операция;
   if (условие)
   {
      // одна или более операция;
      if (условие)
      {
         // одна или более операция;
      }
   }
}
```



Пример:

```
public int godyraboty = 3;

public double zarplata = 1500;

public int pribavka = 0;

if (godyraboty < 5)

{

   if (zarplata < 500)

   {

   pribavka = 100;

   }

   else

   {

   pribavka = 200;

   }

}

else

{

pribavka = 700;

}

Console.WriteLine(“Сумма прибавки: “ + pribavka);
```

Вывод:

Сумма прибавки: 200

Конструкция switch…case

- В случае проверки точного соответствия, для замены нескольких операций if можно использовать switch…case.
- Операция switch…case используется тогда, когда необходимо сравнить переменную с различными значениями.

- **Switch -** Ключевое слово, за которым следует целое выражение, заключенное в круглые скобки.

- **Case -** Ключевое слово, за которым следует уникальная целая константа и двоеточие.

- **Default -** Если ни одно из значений case не совпало со значением выражения в switch, управление программы передается в блок default.

- **Break -** Используется внутри операции switch…case для завершения выполнения последовательности операций.

```
switch (выражение)
    {
        case value1:
            //ряд операторов
            break;
        case value2:
            //ряд операторов
            break;
        .....
        .....
        case valueN:
            //ряд операторов
            break;
        default:
            //ряд операторов по умолчанию
    }
```

Пример:

```
public int day = 5;

switch (day)

{

   case 1:

      Console.WriteLine(“Понедельник”);

      break;

   case 2:

      Console.WriteLine(“Вторник”);
```

```
        break;
    case 3:
        Console.WriteLine(“Среда”);
        break;
    case 4:
        Console.WriteLine(“Четверг”);
        break;
    case 5:
        Console.WriteLine(“Пятница”);
        break;
    case 6:
        Console.WriteLine(“Суббота”);
        break;
    case 7:
        Console.WriteLine(“Воскресенье”);
        break;
    default:
        Console.WriteLine(“Введите число от 1 до 7”);
        break;
}
```

Вывод:

Пятница

Занятие 2 - Циклические конструкции

На втором занятии, **Циклические конструкции**, вы научитесь:

- Описывать циклические конструкции.
- Формулировать синтаксис и работу цикла while.
- Формулировать синтаксис и работу цикла do..while.
- Формулировать синтаксис и работу цикла for.

- Формулировать синтаксис и работу цикла foreach.

Циклические конструкции

- Циклы позволяют несколько раз выполнить один блок операций.

- Циклические конструкции также содержат итераторы.

- C# поддерживает четыре типа циклических конструкций:
  - Цикл while
  - Цикл do..while
  - Цикл for
  - Цикл foreach

Цикл “while”

- Используется для повторного выполнения блока кода до тех пор, пока условие цикла остается истинным.

- Содержит операцию while(начинается с ключевого слова while, за которым следует логическое условие).

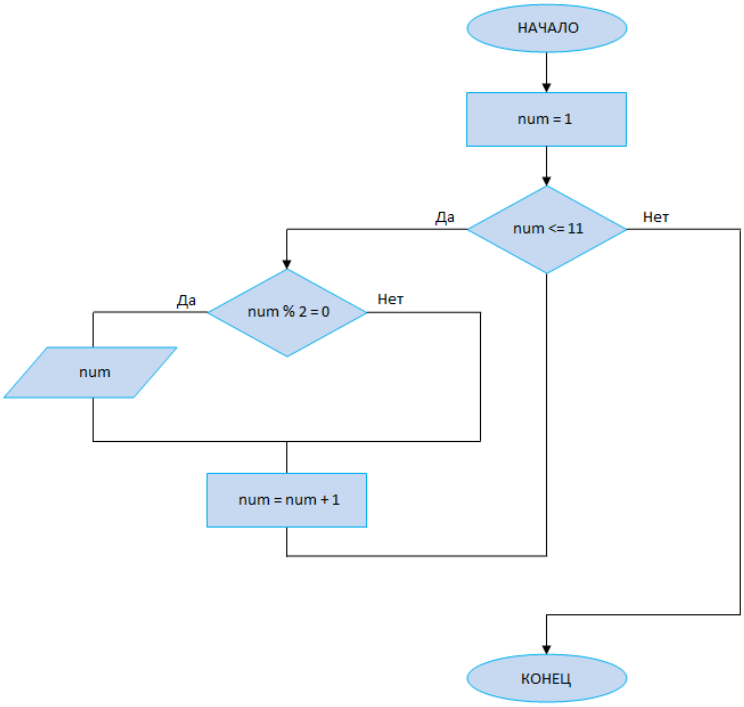
- После каждой итерации управление возвращается в операцию while и условие снова проверяется для следующего раунда выполнения.

Синтаксис:

```
while (условие)

{

   // одна или более операция;

}
```

где,

условие: заданное булево выражение.



Пример:

```
public int num = 1;

Console.WriteLine(“Четные числа”);

while (num <= 11)

{

   if ((num % 2) == 0)

   {

      Console.WriteLine(num);

   }

   num = num + 1;

}
```

Вывод:

Четные числа

2

4

6

8

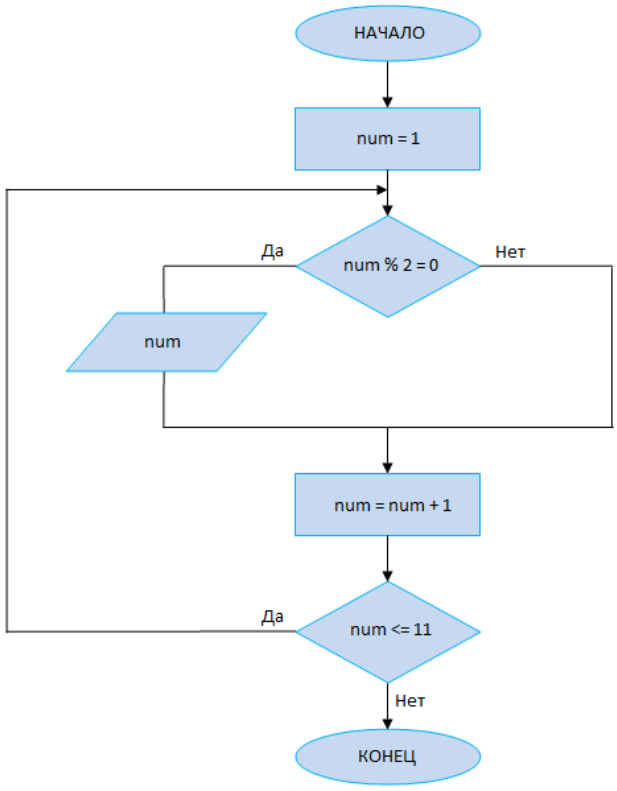
10

Цикл “do-while”

- Цикл do-while похож на цикл while.
- Он всегда выполняется хотя бы раз без проверки условия.
- Цикл начинается с ключевого слова do, за которым следует блок выполняемых операций.
- Операция while с условием располагается в конце этого блока.

Синтаксис:

```
do
{
   // одна или более операция;
} while (условие);
```



Пример:

```
public int num = 1;
   Console.WriteLine(“Четные числа”);
```

```
   do
   {
      if ((num % 2) == 0)
      {
                Console.WriteLine(num);
      }
    num = num + 1;
} while (num <= 11);
```

Вывод:

Четные числа

2

4

6

8

10

Цикл “for”

- Операция for по функционалу похожа на операцию while.
- Операции тела цикла выполняются до тех пор, пока условие истинно.

Синтаксис:

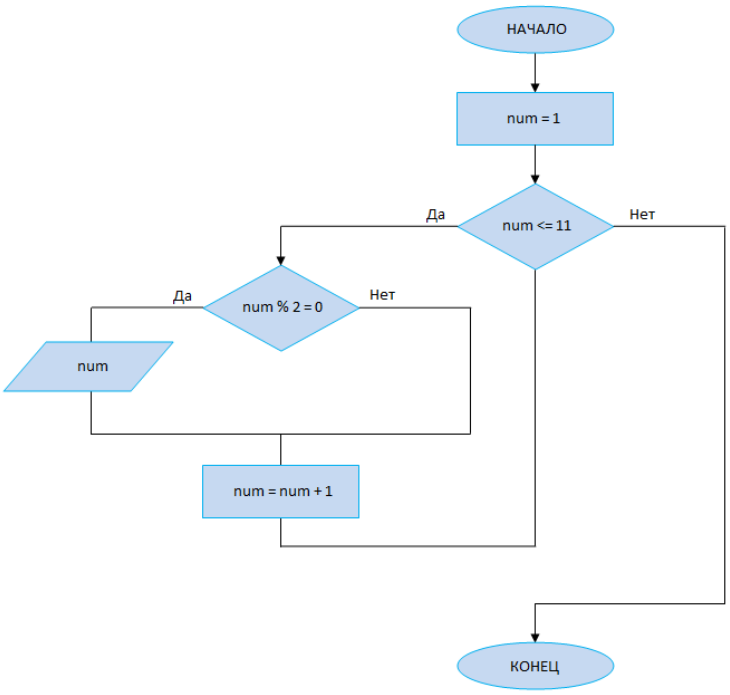
```
for (инициализация; условие; инкремент/декремент)
{
   // одна или более операция;
}
```

где,

инициализация: инициализация переменной (переменных),

которая используется в условии.

условие: включает условие, которое проверяется перед выполнением операций цикла. инкремент/декремент: Содержит операции, которые изменяют значения переменной (переменных) для гарантии, что условие, заданное в секции условий, изменится.



Пример:

```
public int num;
Console.WriteLine(“Четные числа”);
for (num = 1; num <= 11; num++)
{
   if ((num % 2) == 0)
   {
        Console.WriteLine(num);
   }
}
```

Вывод:

Четные числа

2

4

6

8

10

Вложенный цикл “for”

- Содержит несколько операций for.
- Когда один цикл for заключается внутрь другого цикла for, говорят, что цикл вложенный.
- Цикл for, содержащий другой цикл for, называется внешним циклом.
- Внешний цикл for определяет, сколько раз будет вызываться внутренний цикл for.

Пример:

```
public int rows = 2; //указывается число строк

   public int columns = 2; //указывается число столбцов

   for (int i = 0; i < rows; i++)

   {

      for (int j = 0; j < columns; j++)

      {

         Console.Write(“{0} “,i*j);

      }

      Console.WriteLine();

   }
```

Вывод:

0 0

0 1

Цикл “foreach”

- Он просматривает все значения в заданном списке и выполняет блок операций для каждого значения.

- Он начинается с операции foreach, которая позволяет вам задать идентификатор, содержащий список значений.
- Пока выполняется цикл foreach, все элементы, заданные в операции, доступны только для чтения.

Синтаксис:

```
foreach (<тип данных> <идентификатор> in <список>)
{
   // одна или более операция;
}
```

где,

тип данных: Определяет тип данных элементов списка

идентификатор: Имя, подходящее для массива элементов.

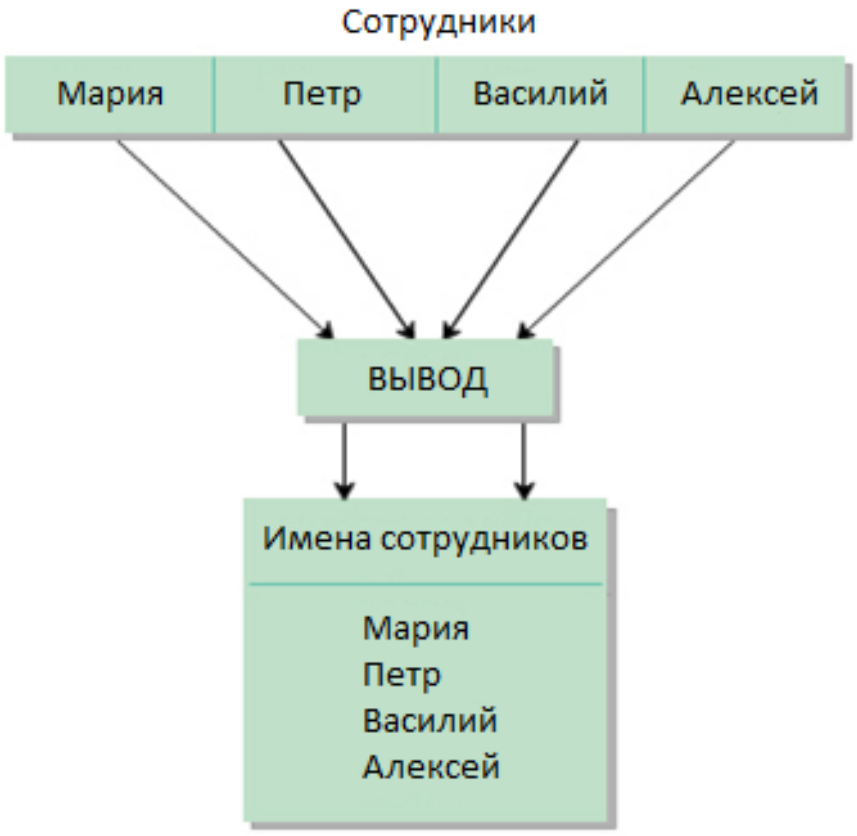
список: Указывает имя списка.

Пример:

```
public string[] employeeNames = { “Мария”, “Петр”, “Василий”, “Алексей” };
Console.WriteLine(“Имена сотрудников”);
foreach (string names in employeeNames)
{
   Console.WriteLine(“{0} “, names);
}
```

Вывод:

```
Имена сотрудников
Мария
Петр
Василий
Алексей
```



Занятие 3 - Операции перехода в C#

На последнем занятии, **Операции перехода в C#**, вы научитесь:

- Описывать операцию jump и ее назначение.
- Объяснять операцию break.
- Описывать операцию continue.
- Описывать операцию goto.
- Объяснять операцию return.

Операции перехода

- Операции перехода используются для передачи управления из одной точки программы в другую.

- Операции перехода безоговорочно переводят управление программы в другое место.

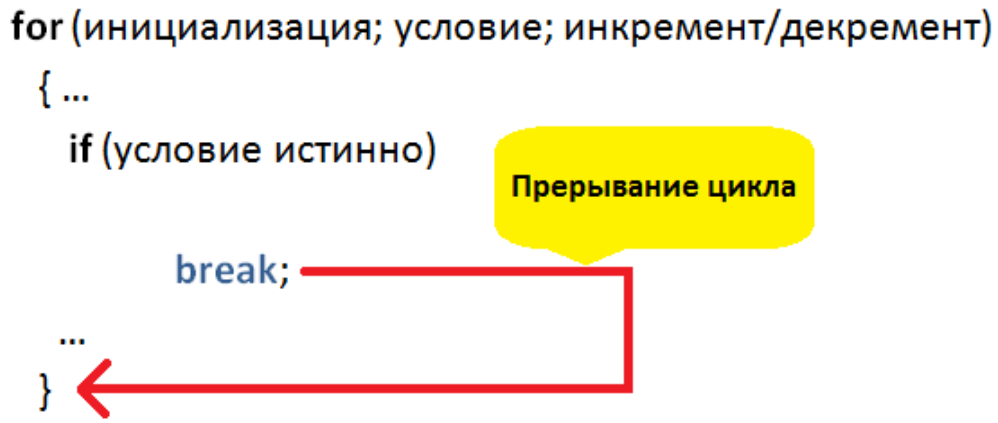
- C# поддерживает четыре типа операций перехода:
  - break
  - continue
  - goto
  - return

Операция “break”

- Используется в конструкциях выбора и циклах.

- Чаще всего используется в конструкции switch...case, в циклах for и while.

- Обозначается ключевым словом break.



Пример:

```
public int numOne = 17;

public int numTwo = 2;

while(numTwo <= numOne-1)

{

   if(numOne % numTwo == 0)

   {

      Console.WriteLine(“Составное число”);

      break;

   }

   numTwo++;

}

if(numTwo == numOne)

{

   Console.WriteLine(“Простое число”);

}
```

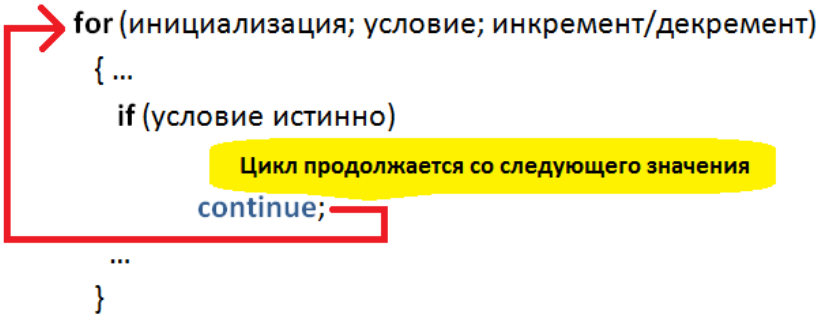
Вывод:

Простое число

Операция “continue”

- Чаще всего используется в циклических конструкциях.

- Обозначается ключевым словом continue.
- Используется для завершения текущей итерации цикла и передачи управления в начало цикла.



Пример:

```
Console.WriteLine(“Четные числа в диапазоне от 1 до 10”);

for (int i=1; i<=10; i++)

{

   if (i % 2 != 0)

   {

      continue;

   }

   Console.Write(i + “ “);

}
```

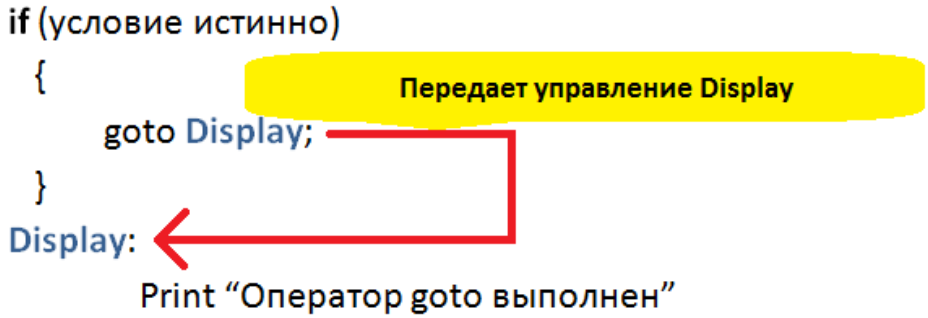
Вывод:

Четные числа в диапазоне от 1 до 10

2 4 6 8 10

Операция “goto”

- Позволяет напрямую выполнить отмеченную операцию или блок операций.
- Обозначается ключевым словом goto.



Пример:

```
public int i = 0;

display:

   Console.WriteLine(“Привет”);

   i++;

   if (i < 5)

   {

      goto display;

   }
```

Вывод:

Привет

Привет

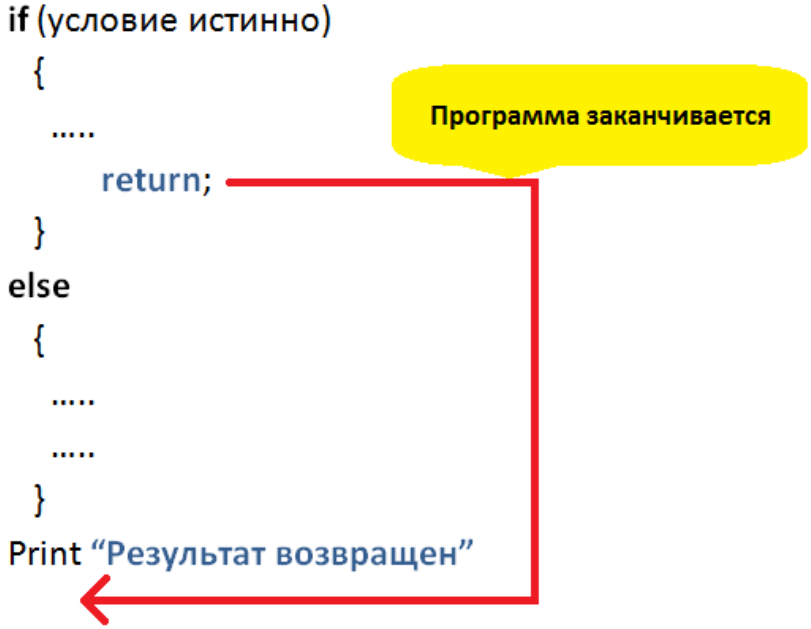
Привет

Привет

Привет

Операция “return”

- Используется для возвращения результата выражения.
- Используется для возвращения управления в метод, из которого текущий метод был вызван.
- Обозначается ключевым словом return.
- Должна быть последней операцией в методе.



Пример:

```
public int godyraboty = 5;

public double zarplata = 1250;

public double pribavka = 0;

if (godyraboty <= 5)

{

   pribavka = 50;

   return;

}

else

{

   pribavka = zarplata * 0.2;

}

Console.WriteLine(“Количество зарплаты: “ + zarplata);

Console.WriteLine(“Количество прибавки: “ + pribavka);
```

Вывод:

Куб числа 23 = 12167

Заключение

- **Конструкции выбора**
  - Это блоки принятия решений, которые выполняют группы операций в зависимости от значения булева условия.

- **Циклические конструкции**
  - Выполняют блок операций несколько раз, в зависимости от заданного условия.

- **Операции перехода в C#**
  - Передают управление в любую отмеченную операцию или блок программы.

Текст презентации к лекции №7,8

Массивы

Обзор модуля:

В этом модуле вы изучите

- Введение в массивы
- Типы массивов
- Класс Array

Занятие 1 – Введение в массивы:

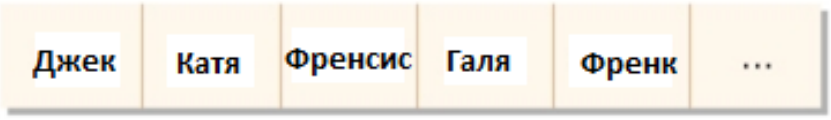
На первом занятии, **Введение в массивы**, вы изучите:

- Определение массива и его назначение.
- Синтаксис объявления массивов.

Назначение:

- Массив - это коллекция связанных значений, размещенных в соседних ячейках памяти, и эти значения адресуются с помощью общего имени массива.

- Это упрощает задачу хранения этих значений.



Массив из 100 имен

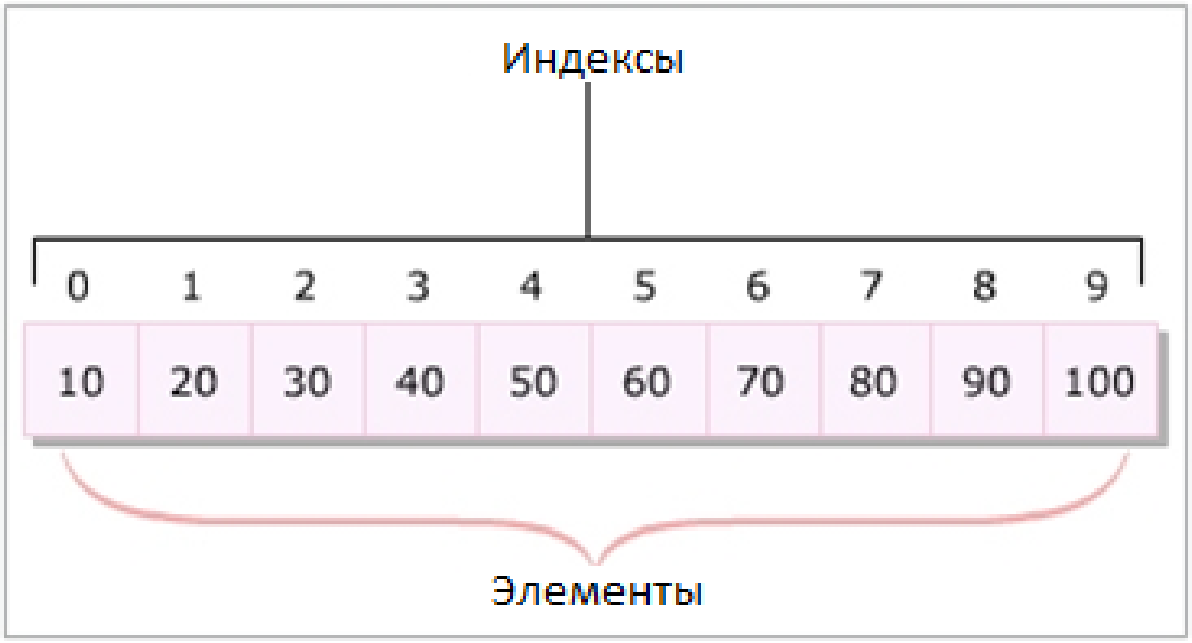
Эффективное использование памяти

```
//Программа для хранения 100 имен студентов
string studentOne = " Джек Андерсон ";
string studentTwo = "  Катя Джонс";
string studentThree = " Френсис Диаз ";
string studentFour = " Галя Даниель ";
string studentFive = "Френк Джеймс";
…
…
… Till 100 variables
```

100 переменных, хранящие имена
Неэффективное использование памяти

Определение:

- Массив всегда хранит значения одного типа данных.
- Каждое значение рассматривается как его элемент.
- C# поддерживает массивы с индексом, который начинается с нуля.
- Такое расположение величин помогает в эффективном хранении, легкой сортировке и простому определению длины данных.



Массивы

Объявление массивов:

- Массивы - это переменные ссылочного типа, создание которых использует два этапа: объявление и выделение памяти.
- Объявление массива указывает тип данных, который он будет хранить и идентификатор.
- Объявление массива не выделяет для него памяти.

Синтаксис:

type[] arrayName;

где,

type: указывает тип данных элементов массива (например, int и char).

arrayName: определяет имя массива.

Инициализация массивов:

- Массив может быть создан с помощью ключевого слова new, а затем инициализирован.

- Массив может быть инициализирован во время своего объявления, в этом случае ключевое слово new не используется.

- Создание и инициализация массива ключевым словом new включает указание размера массива.

- Таблица перечисляет значения по умолчанию для некоторых широко используемых типов данных.

Типы данных	**Значения по умолчанию**
int	0
float	0.0
double	0.0
char	'\0'
string	null

- Следующий код создает массив целых чисел, которых может быть не более пяти.

Синтаксис:

- Следующий код создает массив целых чисел, которых может быть не более пяти.

arrayName = new type[size-value];

- Следующий синтаксис используется для объявления и создания массива в том же выражении с помощью ключевого слова new.

type[] arrayName = new type[size-value];

где,

size-value: Указывает число элементов в массиве.

- Следующий код инициализирует строковый массив, который присваивает имена для заданных значений индекса.

public string[] studNames = new string{“Allan”, “Wilson”, “James”, “Arnold”};

- Следующий код сохраняет строку “Jack” как имя пятого поступившего студента.

type[ ] arrayIdentifier = {val1, val2, val3, ..., valN};

где,

val1: значение первого элемента.

valN: значение n-го элемента.

Пример:

public int[] number = new int[5];

public string[] studNames = new string{“Allan”, “Wilson”, “James”, “Arnold”};

```
class Numbers
{
   static void Main(string[] args)
   {
      int[] count = new int[10];//создание массива
      int counter = 0;
      for(int i = 0; i < 10; i++)
      {
         count[i] = counter++; //присваивание значений      элементам
    Console.WriteLine(“Значение счетчика: “ +          count[i]);
    //вывод значений элементов
      }
   }
}
```

Вывод:

```
The count value is: 0
The count value is: 1
The count value is: 2
The count value is: 3
The count value is: 4
The count value is: 5
```

The count value is: 6

The count value is: 7

The count value is: 8

The count value is: 9

На втором занятии, **Типы массивов**, вы изучите:

- Объяснение одномерных массивов.
- Объяснение многомерных массивов.
- Объяснение неровных массивов и их использование.
- Объяснение, как проходить циклом по массивам, используя foreach- цикл.

Одномерные массивы:

- Элементы одномерного массива сохраняются единой строкой в выделенной памяти.

- Объявление и инициализация одномерных массивов такие же, как и стандартные объявления и инициализация массивов.

- Элементы индексируются от 0 до (n-1), где n - число элементов массива.

Синтаксис:

type[] arrayName; //объявление

arrayName = new type[length]; // создание

где,

type: Тип переменной, сопровождаемый квадратными скобками ([]).

arrayName: определяет имя переменной

length: Указывает число элементов в объявляемом массиве.

new: Создает экземпляр массива.

Пример:

```
class SingleDimensionArray
{
   static void Main(string[] args)
   {
      string[] students = new string[3] {“James”, “Alex”, “Fernando”};
    for (int i=0; i < students.Length; i++)
    {
        Console.WriteLine(students[i]);
    }
   }
}
```

Вывод:

Джеймс

Алекс

Фернандо

Многомерные массивы

- Позволяют сохранять комбинацию значений одного типа в двух и более измерениях.

- Измерения массива представлены строками и столбцами, аналогичными листу Microsoft Excel.

- Есть два типа многомерных массивов:
  - Прямоугольный массив
  - Неровный массив

- **Прямоугольный массив**
  - Это многомерный массив, в котором все указанные измерения имеют постоянные значения.
  - Он всегда имеет одинаковое число столбцов в каждой строке.

- **Неровный массив**
  - Это многомерный массив, в котором одно из указанных измерений может иметь различающиеся значения.
  - Он может иметь неравное число столбцов для каждой строки.

СИНТАКСИС:

type[,] <arrayName>; //объявление

arrayName = new type[value1 , value2]; //инициализация

где,

type: Тип данных, сопровождаемый [].

arrayName: определяет имя массива.

value1: Указывает число строк.

value2: Указывает число столбцов.

ПРИМЕР:

```
class RectangularArray
{
   static void Main (string [] args)
```

```
    {
        int[,] dimension = new int [4, 5];
        int numOne = 0;
        for (int i=0; i<4; i++)
        {
            for (int j=0; j<5; j++)
            {
                    dimension [i, j] = numOne;
                    numOne++;
            }
        }
        for (int i=0; i<4; i++)
        {
            for (int j=0; j<5; j++)
            {
                Console.Write(dimension [i, j] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

ВЫВОД:

0 1 2 3 4

5 6 7 8 9

10 11 12 13 14

15 16 17 18 19

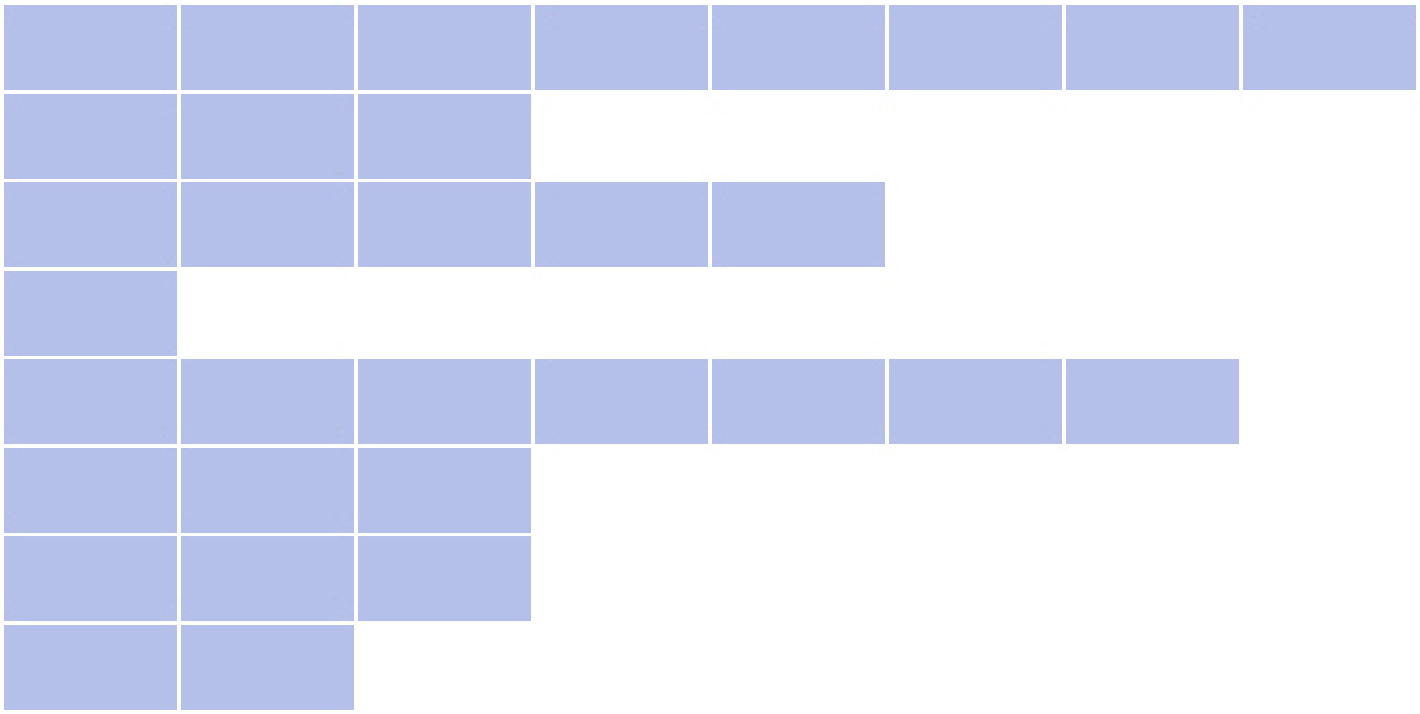
Неровный массив

- Неровный массив - это многомерный массив, выглядящий как массив массивов.

- Он состоит из нескольких массивов, где число элементов в каждом массиве может различаться.

- Он оптимизирует использование памяти и производительность.

Неровный массив:



ПРИМЕР:

```
class JaggedArray
{
   static void Main (string[] args)
   {
      string[][] companies = new string[3][];
      companies[0] = new string[] {“Intel”, “AMD”};
      companies[1] = new string[] {“IBM”, “Microsoft”, “Sun”};
      companies[2] = new string[] {“HP”, “Canon”, “Lexmark”, “Epson”};
      for (int i=0; i<companies.GetLength (0); i++)
      {
         Console.Write(“List of companies in group “ + (i+1) + “:\t”);
         for (int j=0; j<companies[i].GetLength (0); j++)
         {
            Console.Write(companies [i][j] + “ “);
         }
```

```
            Console.WriteLine();
        }
    }
}
```

ВЫВОД:

List of companies in group 1: Intel AMD

List of companies in group 2: IBM Microsoft Sun

List of companies in group 3: HP Canon Lexmark Epson

Использование цикла “foreach” для массивов

- Цикл foreach в C# - это расширение цикла for.
- Цикл используется для осуществления специальных действий над такими коллекциями, как массивы.
- Цикл читает каждый элемент в указанном массиве.
- Он позволяет запускать блок кода для каждого элемента массива.

СИНТАКСИС:

```
foreach(type<identifier> in <list>)
{
   // statements
```

где,

type: Тип переменной.

identifier: Имя переменной.

list: Имя переменной массива.

ПРИМЕР:

```
class Students
{
   static void Main(string[] args)
   {
      string[] studentNames = new string[3] { “Эшли”,      “Джо”, “Майкл”};
      foreach (string studName in studentNames)
      {
         Console.WriteLine(“Поздравляю!! “ +            studName + “ вы получили дополнительный отпуск”);
      }
   }
}
```

ВЫВОД:

Поздравляю!! Эшли вы получили дополнительный отпуск

Поздравляю!! Джо вы получили дополнительный отпуск

Поздравляю!! Майкл вы получили дополнительный отпуск

Занятие 3 - Класс Array

На последнем занятии, **Array класс** , вы изучите:

- Описание класса Array и его назначение.
- Список обычно используемых свойств и методов класса Array.
- Объяснение, как конструировать массив с помощью класса Array.

Класс “Array”

- Это встроенный класс в пространстве имен System и базовый класс для всех массивов в C#.

- Он обеспечивает методы для таких различных задач, как:
  - создание
  - поиск
  - копирование
  - сортировка массивов

Свойства и методы:

- Класс Array состоит из системно-определенных свойств и методов, используемых для создания и управления массивами в C#.

- Свойства также называются системными свойствами класса Array.

- **Свойства**
  - Свойства класса Array позволяют изменять элементы, объявленные в массиве.

  - Таблица отображает свойства класса Array.

Свойства	Описания
IsFixedSize	Возвращает логическое значение, указывающее, имеет ли массив заданный размер или нет. Значение по умолчанию - true.
IsReadOnly	Возвращает логическое значение, указывающее, доступен ли массив только по чтению или нет. Значение по умолчанию - false.
IsSynchronized	Возвращает логическое значение, указывающее, способен ли массив поддерживать одновременно несколько выполняющихся потоков. Значение по умолчанию - false.
Length	Возвращает 32-битное целое, указывающее общее число элементов массива.
LongLength	Возвращает 64-битное целое, указывающее общее число элементов массива.
Rank	Возвращает целое число, указывающее ранг, означающий число измерений массива.
SyncRoot	Возвращает объект, использующийся для синхронизации доступа к массиву.

- **Методы**
  - Класс Array позволяет очищать, копировать, искать и сортировать элементы, объявленные в массиве.

  - Таблица отображает часто используемые методы класса Array.

Методы	**Описания**
Clear	Удаляет все элементы в массиве и устанавливает его размер в 0.
CopyTo	Копирует все элементы текущего одномерного массива в другой одномерный массив, начиная с указанной позиции индекса.
GetLength	Возвращает число элементов в массиве.
GetLowerBound	Возвращает нижнюю границу массива.
GetUpperBound	Возвращает верхнюю границу массива.
Initialize	Инициализирует каждый элемент массива вызовом конструктора по умолчанию класса Array.

Sort	Сортирует элементы одномерного массива.
SetValue	Устанавливает указанное значение в заданной индексом позиции массива.
GetValue	Возвращает значение в заданной индексом позиции массива.

Использование класса “Array”

- Класс Array позволяет создавать массивы с помощью метода CreateInstance().
- Он может использоваться с различными параметрами для создания одномерных и многомерных массивов.

СИНТАКСИС:

public static Array CreateInstance(Type elementType, int length1, int length2)

где,

Array: Возвращает ссылку на созданный массив.

Type: Использует оператор typeof для явного приведения типов.

elementType: Результирующий тип после приведения.

Length: Указывает длину массива.

ПРИМЕР:

class Subjects

```
{
   static void Main(string [] args)
   {
      Array objArray = Array.CreateInstance(typeof                (string), 5);
      objArray.SetValue("Маркетинг", 0);
      objArray.SetValue("Финансы", 1);
      objArray.SetValue("Человеческий ресурс", 2);
      objArray.SetValue("Информациооные технологии", 3);
      objArray.SetValue("Бизнес-администрирование", 4);
      for (int i = 0; i <= objArray.GetUpperBound(0); i++)
      {
         Console.WriteLine(objArray.GetValue(i));
      }
   }
}
```

Резюме

- **Массивы**
  - Массивы - коллекции значений одного типа данных.
- **Типы массивов**
  - Есть два типа массивов в C# - одномерные и многомерные.
- **Класс Array**
  - Класс Array обеспечивает методы и свойства для создания, поиска и сортировки массивов.