МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ГЛАВНОЕ УПРАВЛЕНИЕ ПО ОБРАЗОВАНИЮ

МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

|  |  |
| --- | --- |
| Специальность | 2– 40 01 01 |
| Учебная группа | ПО-455 |
|  |  |

Учебная дисциплина Конструирование программ

и языки программирования

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

**РАЗРАБОТКА ЛИНЕЙНЫХ ПРОГРАММ**

Выполнил Савич А.О.

Проверил Карманов А.В.

2021

1. Цель работы

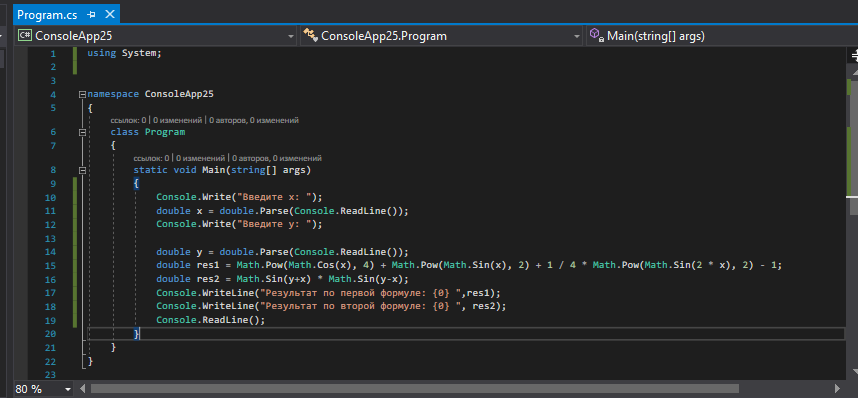
Сформировать умения разрабатывать линейные программы с объявлением переменных и констант, строить выражения на основе приоритета операций

1. Индивидуальное задание

Вариант 8

Напишите программу для расчета по двум формулам. Предварительно подготовьте тестовые примеры по второй формуле с помощью калькулятора (результат вычисления по первой формуле должен совпадать со второй) Значения выражения вывести - в формате с плавающей точкой, ограничить количество знаков после запятой до трех - в научной нотации - с применением пользовательского шаблона, в целой и дробной части не менее 5 знаков 10 - с применением пользовательского шаблона в целой и дробной частях не менее 5 знаков, если они не являются нулями - используя операцию конкатенации строк Математические функции реализованы как методы класса Math и представлены в приложении А





1. Контрольные вопросы

1 Приведите синтаксис и опишите алгоритм выполнения простой операции присваивания

Операции присвоения устанавливают значение. В операциях присвоения участвуют два операнда, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную

Как и во многих других языках программирования, в C# имеется базовая операция присваивания **=**, которая присвоивает значение правого операнда левому операнду:

int number = 23;

Здесь переменной number присваивается число 23. Переменная number представляет левый операнд, которому присваивается значение правого операнда, то есть числа 23.

Также можно выполнять множественно присвоение сразу нескольких переменным одновременно:

int a, b, c;

a = b = c = 34;

Стоит отметить, что операции присвоения имеют низкий приоритет. И вначале будет вычисляться значение правого операнда и только потом будет идти присвоение этого значения левому операнду. Например:

int a, b, c;

a = b = c = 34 \* 2 / 4; // 17

Сначала будет вычисляться выражение 34 \* 2 / 4, затем полученное значение будет присвоено переменным.

Кроме базовой операции присвоения в C# есть еще ряд операций:

**+=**: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: выражение **A += B** равнозначно выражению **A = A + B**

**-=**: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: **A -= B** эквивалентно **A = A - B**

**\*=**: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: **A \*= B** эквивалентно **A = A \* B**

**/=**: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: **A /= B** эквивалентно **A = A / B**

**%=**: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: **A %= B** эквивалентно **A = A % B**

**&=**: присваивание после поразрядной конъюнкции. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: **A &= B** эквивалентно **A = A & B**

**|=**: присваивание после поразрядной дизъюнкции. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: **A |= B** эквивалентно **A = A | B**

**^=**: присваивание после операции исключающего ИЛИ. Присваивает левому операнду результат операции исключающего ИЛИ его битового представления с битовым представлением правого операнда: **A ^= B** эквивалентно **A = A ^ B**

**<<=**: присваивание после сдвига разрядов влево. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: **A <<= B** эквивалентно **A = A << B**

**>>=**: присваивание после сдвига разрядов вправо. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: **A >>= B** эквивалентно **A = A >> B**

2 Приведите синтаксис и опишите алгоритм выполнения бинарной операции присваивания

Операции присвоения устанавливают значение. В операциях присвоения участвуют два операнда, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную

Как и во многих других языках программирования, в C# имеется базовая операция присваивания **=**, которая присвоивает значение правого операнда левому операнду:

int number = 23;

Здесь переменной number присваивается число 23. Переменная number представляет левый операнд, которому присваивается значение правого операнда, то есть числа 23.

Также можно выполнять множественно присвоение сразу нескольких переменным одновременно:

int a, b, c;

a = b = c = 34;

Стоит отметить, что операции присвоения имеют низкий приоритет. И вначале будет вычисляться значение правого операнда и только потом будет идти присвоение этого значения левому операнду. Например:

int a, b, c;

a = b = c = 34 \* 2 / 4; // 17

Сначала будет вычисляться выражение 34 \* 2 / 4, затем полученное значение будет присвоено переменным.

Кроме базовой операции присвоения в C# есть еще ряд операций:

**+=**: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: выражение **A += B** равнозначно выражению **A = A + B**

**-=**: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: **A -= B** эквивалентно **A = A - B**

**\*=**: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: **A \*= B** эквивалентно **A = A \* B**

**/=**: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: **A /= B** эквивалентно **A = A / B**

**%=**: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: **A %= B** эквивалентно **A = A % B**

**&=**: присваивание после поразрядной конъюнкции. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: **A &= B** эквивалентно **A = A & B**

**|=**: присваивание после поразрядной дизъюнкции. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: **A |= B** эквивалентно **A = A | B**

**^=**: присваивание после операции исключающего ИЛИ. Присваивает левому операнду результат операции исключающего ИЛИ его битового представления с битовым представлением правого операнда: **A ^= B** эквивалентно **A = A ^ B**

**<<=**: присваивание после сдвига разрядов влево. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: **A <<= B** эквивалентно **A = A << B**

**>>=**: присваивание после сдвига разрядов вправо. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: **A >>= B** эквивалентно **A = A >> B**

3 Перечислите типы данных языка

В языке C# есть следующие примитивные типы данных:

* **bool**: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean

|  |  |
| --- | --- |
| 1  2 | bool alive = true;  bool isDead = false; |

* **byte**: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte

|  |  |
| --- | --- |
| 1  2 | byte bit1 = 1;  byte bit2 = 102; |

* **sbyte**: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte

|  |  |
| --- | --- |
| 1  2 | sbyte bit1 = -101;  sbyte bit2 = 102; |

* **short**: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16

|  |  |
| --- | --- |
| 1  2 | short n1 = 1;  short n2 = 102; |

* **ushort**: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16

|  |  |
| --- | --- |
| 1  2 | ushort n1 = 1;  ushort n2 = 102; |

* **int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

|  |  |
| --- | --- |
| 1  2  3 | int a = 10;  int b = 0b101;  // бинарная форма b =5  int c = 0xFF;   // шестнадцатеричная форма c = 255 |

* **uint**: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32

|  |  |
| --- | --- |
| 1  2  3 | uint a = 10;  uint b = 0b101;  uint c = 0xFF; |

* **long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64

|  |  |
| --- | --- |
| 1  2  3 | long a = -10;  long b = 0b101;  long c = 0xFF; |

* **ulong**: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64

|  |  |
| --- | --- |
| 1  2  3 | ulong a = 10;  ulong b = 0b101;  ulong c = 0xFF; |

* **float**: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта. Представлен системным типом System.Single
* **double**: хранит число с плавающей точкой от ±5.0\*10-324 до ±1.7\*10308 и занимает 8 байта. Представлен системным типом System.Double
* **decimal**: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0\*10-28 до ±7.9228\*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal
* **char**: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

|  |  |
| --- | --- |
| 1  2  3 | char a = 'A';  char b = '\x5A';  char c = '\u0420'; |

* **string**: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы.

|  |  |
| --- | --- |
| 1  2 | string hello = "Hello";  string word = "world"; |

* **object**: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

|  |  |
| --- | --- |
| 1  2  3 | object a = 22;  object b = 3.14;  object c = "hello code"; |

4 Опишите синтаксис метода Read

тип переменной имя = Console.Read();

5 Опишите синтаксис метода ReadLine

тип переменной имя = Console.ReadLine();

6 Опишите синтаксис метода Write

тип переменной имя = Console.Write();

7 Опишите синтаксис метода WriteLine

тип переменной имя = Console.WriteLine();

8 Опишите применение пользовательский шаблонов в методе WriteLine

Возможность использовать сопоставление шаблонов впервые появилась в C# 7.0. С тех пор в каждой новой основной версии C# возможности сопоставления шаблонов расширяются. Сопоставление шаблонов поддерживают следующие выражения и операторы C#:

Выражение is

Оператор switch

Выражение switch (появилось в C# 8.0)

В этих конструкциях можно сравнить входное выражение с любым из следующих шаблонов:

Шаблон объявления: для проверки типа выражения в среде выполнения и, в случае равенства, присвоения объявленной переменной результата этого выражения. Впервые появился в C# 7.0.

Шаблон типа: для проверки типа выражения в среде выполнения. Впервые появился в C# 9.0.

Шаблон константы: для проверки того, равен ли результат выражения указанной константе. Впервые появился в C# 7.0.

Реляционные шаблоны: для сравнения результата выражения с заданной константой. Впервые появился в C# 9.0.

Логические шаблоны: для проверки того, соответствует ли выражение логической комбинации шаблонов. Впервые появился в C# 9.0.

Шаблон свойства: для проверки того, соответствуют ли свойства или поля выражения вложенным шаблонам. Впервые появился в C# 8.0.

Позиционный шаблон: для деконструкции результата выражения и проверки того, соответствуют ли результирующие значения вложенным шаблонам. Впервые появился в C# 8.0.

Шаблон var: для сравнения любых выражений и присваивания результата сравнения объявленной переменной. Впервые появился в C# 7.0.

Шаблон пустой переменной: для сравнения любых выражений. Впервые появился в C# 8.0.

Логические шаблоны, шаблоны свойств и позиционные шаблоны являются рекурсивными шаблонами. То есть, они могут содержать вложенные шаблоны.

Пример использования этих шаблонов для создания управляемого данными алгоритма можно посмотреть в разделе Учебник: использование сопоставления шаблонов для создания управляемых типами и управляемых данными алгоритмов.

Шаблоны объявления и шаблоны типов

Шаблоны объявления и шаблоны типов используются для проверки того, совместим ли с указанным типом тип определенного выражения в среде выполнения. С помощью шаблона объявления можно также объявить новую локальную переменную. Если шаблон объявления соответствует выражению, этой переменной присваивается результат преобразованного выражения, как показано в следующем примере:

C#Копировать

object greeting = "Hello, World!";

if (greeting is string message)

{

Console.WriteLine(message.ToLower()); // output: hello, world!

}

Начиная с C# 7.0, шаблон объявления с типом T соответствует выражению в том случае, если результат выражения имеет значение, отличное от NULL, и выполняется любое из следующих условий:

Тип результата выражения в среде выполнения — T.

Тип результата выражения в среде выполнения является производным от типа T, реализует интерфейс T или существует другое неявное преобразование ссылок из него в T. В следующем примере показаны два случая, когда данное условие истинно:

C#Копировать

var numbers = new int[] { 10, 20, 30 };

Console.WriteLine(GetSourceLabel(numbers)); // output: 1

var letters = new List<char> { 'a', 'b', 'c', 'd' };

Console.WriteLine(GetSourceLabel(letters)); // output: 2

static int GetSourceLabel<T>(IEnumerable<T> source) => source switch

{

Array array => 1,

ICollection<T> collection => 2,

\_ => 3,

};

В предыдущем примере при первом вызове метода GetSourceLabel первый шаблон соответствует значению аргумента, так как тип этого аргумента (int[]) в среде выполнения является производным от типа Array. При втором вызове метода GetSourceLabel тип аргумента в среде выполнения (List<T>) не является производным от типа Array, но реализует интерфейс ICollection<T>.

Тип результата выражения в среде выполнения является типом, допускающим значение NULL, и имеющим базовый тип T.

Существует упаковка-преобразование или распаковка-преобразование из типа результата выражения в среде выполнения в тип T.

В следующем примере показаны два последних условия:

C#Копировать

int? xNullable = 7;

int y = 23;

object yBoxed = y;

if (xNullable is int a && yBoxed is int b)

{

Console.WriteLine(a + b); // output: 30

}

Если требуется проверить только тип выражения, можно вместо имени переменной использовать пустую переменную \_, как показано в следующем примере:

C#Копировать

public abstract class Vehicle {}

public class Car : Vehicle {}

public class Truck : Vehicle {}

public static class TollCalculator

{

public static decimal CalculateToll(this Vehicle vehicle) => vehicle switch

{

Car \_ => 2.00m,

Truck \_ => 7.50m,

null => throw new ArgumentNullException(nameof(vehicle)),

\_ => throw new ArgumentException("Unknown type of a vehicle", nameof(vehicle)),

};

}

Начиная с C# 9.0, для этой цели можно использовать шаблон типа, как показано в следующем примере:

C#Копировать

public static decimal CalculateToll(this Vehicle vehicle) => vehicle switch

{

Car => 2.00m,

Truck => 7.50m,

null => throw new ArgumentNullException(nameof(vehicle)),

\_ => throw new ArgumentException("Unknown type of a vehicle", nameof(vehicle)),

};

Как и шаблон объявления, шаблон типа соответствует выражению, если результат выражения не равен NULL, а его тип в среде выполнения удовлетворяет любому из указанных выше условий.

Дополнительные сведения см. в разделах Шаблон объявления и Шаблон типа в примечаниях к предлагаемой функции.

Шаблон константы

Начиная с C# 7.0, вы можете использовать шаблон константы для проверки того, равен ли результат выражения заданной константе, как показано в следующем примере:

C#Копировать

public static decimal GetGroupTicketPrice(int visitorCount) => visitorCount switch

{

1 => 12.0m,

2 => 20.0m,

3 => 27.0m,

4 => 32.0m,

0 => 0.0m,

\_ => throw new ArgumentException($"Not supported number of visitors: {visitorCount}", nameof(visitorCount)),

};

В шаблоне константы можно использовать любое константное выражение, например:

целочисленный литерал или литерал с плавающей точкой;

символьный или строковый литерал;

логическое значение true или false;

значение типа enum;

имя объявленного поля с модификатором const или локального элемента.

null

Используйте шаблон константы для проверки на null, как показано в следующем примере:

C#Копировать

if (input is null)

{

return;

}

Компилятор гарантирует, что при вычислении выражения x is null не будет вызван перегруженный пользователем оператор равенства ==.

Начиная с C# 9.0, вы можете использовать шаблон константы null с отрицанием для проверки неравенства значению NULL, как показано в следующем примере:

C#Копировать

if (input is not null)

{

// ...

}

Дополнительные сведения см. в разделе Шаблон константы в примечании к предлагаемой функции.

Реляционные шаблоны

Начиная с C# 9.0, вы можете использовать реляционный шаблон для сравнения результата выражения с константой, как показано в следующем примере:

C#Копировать

Console.WriteLine(Classify(13)); // output: Too high

Console.WriteLine(Classify(double.NaN)); // output: Unknown

Console.WriteLine(Classify(2.4)); // output: Acceptable

static string Classify(double measurement) => measurement switch

{

< -4.0 => "Too low",

> 10.0 => "Too high",

double.NaN => "Unknown",

\_ => "Acceptable",

};

В реляционном шаблоне можно использовать любые операторы отношений — <, >, <=, >=. Правая часть реляционного шаблона должна быть константным выражением. Константное выражение может быть целым числом, числом с плавающей точкой, символом или иметь тип enum.

Чтобы проверить, находится ли результат выражения в определенном диапазоне, сопоставьте его с шаблоном конъюнкции (and), как показано в следующем примере:

C#Копировать

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 3, 14))); // output: spring

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 7, 19))); // output: summer

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 2, 17))); // output: winter

static string GetCalendarSeason(DateTime date) => date.Month switch

{

>= 3 and < 6 => "spring",

>= 6 and < 9 => "summer",

>= 9 and < 12 => "autumn",

12 or (>= 1 and < 3) => "winter",

\_ => throw new ArgumentOutOfRangeException(nameof(date), $"Date with unexpected month: {date.Month}."),

};

Если результат выражения — null или его не удается преобразовать в тип константы с помощью преобразования, допускающего значение NULL, или распаковки-преобразования, то реляционный шаблон не соответствует выражению.

Дополнительные сведения см. в разделе Реляционные шаблоны в примечании к предлагаемой функции.

Логические шаблоны

Начиная с C# 9.0, вы можете использовать блоки объединения not, and и or для создания следующих логических шаблонов:

Шаблон отрицания not, который соответствует выражению, если шаблон с отрицанием не соответствует выражению. В следующем примере показано, как можно инвертировать шаблон константы null, чтобы проверить, имеет ли выражение значение, отличное от NULL:

C#Копировать

if (input is not null)

{

// ...

}

Шаблон конъюнкции and, который соответствует выражению, если оба шаблона соответствуют этому выражению. В следующем примере показано, как можно объединить реляционные шаблоны, чтобы проверить, находится ли значение в определенном диапазоне:

C#Копировать

Console.WriteLine(Classify(13)); // output: High

Console.WriteLine(Classify(-100)); // output: Too low

Console.WriteLine(Classify(5.7)); // output: Acceptable

static string Classify(double measurement) => measurement switch

{

< -40.0 => "Too low",

>= -40.0 and < 0 => "Low",

>= 0 and < 10.0 => "Acceptable",

>= 10.0 and < 20.0 => "High",

>= 20.0 => "Too high",

double.NaN => "Unknown",

};

Шаблон дизъюнкции or, который соответствует выражению, если любой из шаблонов соответствует данному выражению, как показано в следующем примере:

C#Копировать

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 1, 19))); // output: winter

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 10, 9))); // output: autumn

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 5, 11))); // output: spring

static string GetCalendarSeason(DateTime date) => date.Month switch

{

3 or 4 or 5 => "spring",

6 or 7 or 8 => "summer",

9 or 10 or 11 => "autumn",

12 or 1 or 2 => "winter",

\_ => throw new ArgumentOutOfRangeException(nameof(date), $"Date with unexpected month: {date.Month}."),

};

Как показано в предыдущем примере, блоки объединения в шаблоне можно использовать многократно.

Блок объединения шаблонов and имеет более высокий приоритет, чем or. Чтобы явно задать приоритет, используйте круглые скобки, как показано в следующем примере:

C#Копировать

static bool IsLetter(char c) => c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z');

**Примечание**

Порядок проверки шаблонов не определен. Во время выполнения можно сначала проверить правые вложенные шаблоны шаблонов or и and.

Дополнительные сведения см. в разделе Блоки объединения шаблонов в примечании к предлагаемой функции.

Шаблон свойства

Начиная с C# 8.0, вы можете использовать шаблон свойства для сопоставления свойств или полей выражения с вложенными шаблонами, как показано в следующем примере:

C#Копировать

static bool IsConferenceDay(DateTime date) => date is { Year: 2020, Month: 5, Day: 19 or 20 or 21 };

Шаблон свойства соответствует выражению, если результат выражения не равен NULL, а каждый вложенный шаблон соответствует соответствующему свойству или полю результата выражения.

Вы также можете добавить проверку типа среды выполнения и объявление переменной в шаблон свойства, как показано в следующем примере:

C#Копировать

Console.WriteLine(TakeFive("Hello, world!")); // output: Hello

Console.WriteLine(TakeFive("Hi!")); // output: Hi!

Console.WriteLine(TakeFive(new[] { '1', '2', '3', '4', '5', '6', '7' })); // output: 12345

Console.WriteLine(TakeFive(new[] { 'a', 'b', 'c' })); // output: abc

static string TakeFive(object input) => input switch

{

string { Length: >= 5 } s => s.Substring(0, 5),

string s => s,

ICollection<char> { Count: >= 5 } symbols => new string(symbols.Take(5).ToArray()),

ICollection<char> symbols => new string(symbols.ToArray()),

null => throw new ArgumentNullException(nameof(input)),

\_ => throw new ArgumentException("Not supported input type."),

};

Шаблон свойства является рекурсивным шаблоном. Это значит, что любой шаблон можно использовать как вложенный шаблон. Используйте шаблон свойства для сопоставления элементов данных с вложенными шаблонами, как показано в следующем примере:

C#Копировать

public record Point(int X, int Y);

public record Segment(Point Start, Point End);

static bool IsAnyEndOnXAxis(Segment segment) =>

segment is { Start: { Y: 0 } } or { End: { Y: 0 } };

В предыдущем примере используются две возможности, доступные в C# 9.0 и более поздних версиях языка: блок объединения шаблонов orи типы записей.

Начиная с C# 10 можно ссылаться на вложенные свойства или поля в шаблоне свойства. Например, можно выполнить рефакторинг метода из предыдущего примера в следующий эквивалентный код:

C#Копировать

static bool IsAnyEndOnXAxis(Segment segment) =>

segment is { Start.Y: 0 } or { End.Y: 0 };

Дополнительные сведения см. в разделе Шаблон свойства в примечании к предлагаемой функции и примечание к предлагаемой функции Расширенные шаблоны свойств.

Позиционный шаблон

Начиная с C# 8.0, вы можете использовать позиционный шаблон для деконструкции результата выражения и сравнения результирующих значений с соответствующими вложенными шаблонами, как показано в следующем примере:

C#Копировать

public readonly struct Point

{

public int X { get; }

public int Y { get; }

public Point(int x, int y) => (X, Y) = (x, y);

public void Deconstruct(out int x, out int y) => (x, y) = (X, Y);

}

static string Classify(Point point) => point switch

{

(0, 0) => "Origin",

(1, 0) => "positive X basis end",

(0, 1) => "positive Y basis end",

\_ => "Just a point",

};

В предыдущем примере тип выражения содержит метод Deconstruct, используемый для деконструкции результата выражения. Можно также сопоставлять выражения кортежных типов с позиционными шаблонами. Таким образом можно сопоставить несколько входных значений с различными шаблонами, как показано в следующем примере:

C#Копировать

static decimal GetGroupTicketPriceDiscount(int groupSize, DateTime visitDate)

=> (groupSize, visitDate.DayOfWeek) switch

{

(<= 0, \_) => throw new ArgumentException("Group size must be positive."),

(\_, DayOfWeek.Saturday or DayOfWeek.Sunday) => 0.0m,

(>= 5 and < 10, DayOfWeek.Monday) => 20.0m,

(>= 10, DayOfWeek.Monday) => 30.0m,

(>= 5 and < 10, \_) => 12.0m,

(>= 10, \_) => 15.0m,

\_ => 0.0m,

};

В предыдущем примере используются реляционные и логические шаблоны, доступные в C# 9.0 и более поздних версиях языка.

В позиционном шаблоне можно использовать имена элементов кортежа и параметры метода Deconstruct, как показано в следующем примере:

C#Копировать

var numbers = new List<int> { 1, 2, 3 };

if (SumAndCount(numbers) is (Sum: var sum, Count: > 0))

{

Console.WriteLine($"Sum of [{string.Join(" ", numbers)}] is {sum}"); // output: Sum of [1 2 3] is 6

}

static (double Sum, int Count) SumAndCount(IEnumerable<int> numbers)

{

int sum = 0;

int count = 0;

foreach (int number in numbers)

{

sum += number;

count++;

}

return (sum, count);

}

Можно также расширить позиционный шаблон одним из следующих способов:

Добавьте проверку типа среды выполнения и объявление переменной, как показано в следующем примере:

C#Копировать

public record Point2D(int X, int Y);

public record Point3D(int X, int Y, int Z);

static string PrintIfAllCoordinatesArePositive(object point) => point switch

{

Point2D (> 0, > 0) p => p.ToString(),

Point3D (> 0, > 0, > 0) p => p.ToString(),

\_ => string.Empty,

};

В предыдущем примере используются позиционные записи, которые неявно обеспечивают выполнение метода Deconstruct.

Используйте шаблон свойства в позиционном шаблоне, как показано в следующем примере:

C#Копировать

public record WeightedPoint(int X, int Y)

{

public double Weight { get; set; }

}

static bool IsInDomain(WeightedPoint point) => point is (>= 0, >= 0) { Weight: >= 0.0 };

Можно объединить два предыдущих варианта, как показано в следующем примере:

C#Копировать

if (input is WeightedPoint (> 0, > 0) { Weight: > 0.0 } p)

{

// ..

}

Позиционный шаблон является рекурсивным шаблоном. Это значит, что любой шаблон можно использовать как вложенный шаблон.

Дополнительные сведения см. в разделе Позиционный шаблон в примечании к предлагаемой функции.

Шаблон var

Начиная с C# 7.0, шаблон var можно использовать для сопоставления любого выражения, включая null, и присвоения результата сопоставления новой локальной переменной, как показано в следующем примере:

C#Копировать

static bool IsAcceptable(int id, int absLimit) =>

SimulateDataFetch(id) is var results

&& results.Min() >= -absLimit

&& results.Max() <= absLimit;

static int[] SimulateDataFetch(int id)

{

var rand = new Random();

return Enumerable

.Range(start: 0, count: 5)

.Select(s => rand.Next(minValue: -10, maxValue: 11))

.ToArray();

}

Шаблон var полезно использовать, если в логическом выражении вам требуется временная переменная для хранения результатов промежуточных вычислений. Шаблон var также можно использовать, если требуется реализовать дополнительные проверки в условиях регистра when выражения или оператора switch, как показано в следующем примере:

C#Копировать

public record Point(int X, int Y);

static Point Transform(Point point) => point switch

{

var (x, y) when x < y => new Point(-x, y),

var (x, y) when x > y => new Point(x, -y),

var (x, y) => new Point(x, y),

};

static void TestTransform()

{

Console.WriteLine(Transform(new Point(1, 2))); // output: Point { X = -1, Y = 2 }

Console.WriteLine(Transform(new Point(5, 2))); // output: Point { X = 5, Y = -2 }

}

В предыдущем примере шаблон var (x, y) эквивалентен позиционному шаблону (var x, var y).

В шаблоне var тип объявленной переменной равен установленному во время компиляции типу выражения, сопоставляемого с данным шаблоном.

Дополнительные сведения см. в разделе Шаблон var в примечании к предлагаемой функции.

Шаблон пустой переменной

Начиная с C# 8.0, шаблон пустой переменной \_ используется для сопоставления любых выражений, включая null, как показано в следующем примере:

C#Копировать

Console.WriteLine(GetDiscountInPercent(DayOfWeek.Friday)); // output: 5.0

Console.WriteLine(GetDiscountInPercent(null)); // output: 0.0

Console.WriteLine(GetDiscountInPercent((DayOfWeek)10)); // output: 0.0

static decimal GetDiscountInPercent(DayOfWeek? dayOfWeek) => dayOfWeek switch

{

DayOfWeek.Monday => 0.5m,

DayOfWeek.Tuesday => 12.5m,

DayOfWeek.Wednesday => 7.5m,

DayOfWeek.Thursday => 12.5m,

DayOfWeek.Friday => 5.0m,

DayOfWeek.Saturday => 2.5m,

DayOfWeek.Sunday => 2.0m,

\_ => 0.0m,

};

В предыдущем примере шаблон пустой переменной используется для обработки значения null и любых целочисленных значений, которые не соответствуют имеющимся членам перечисления DayOfWeek. Благодаря этому гарантируется, что выражение switch в приведенном примере сможет обработать все возможные входные значения. Если в выражении switch не используется шаблон пустой переменной и при этом ни один из шаблонов выражения не соответствует входным данным, среда выполнения генерирует исключение. Если выражение switch не обрабатывает все возможные входные значения, компилятор генерирует предупреждение.

Шаблон пустой переменной не может быть шаблоном в выражении is или операторе switch. В этих случаях для сопоставления выражений используйте шаблон var с пустой переменной: var \_.

Дополнительные сведения см. в разделе Шаблон пустой переменной в примечании к предлагаемой функции.

Шаблон в круглых скобках

Начиная с C# 9.0, вы можете использовать круглые скобки вокруг любого шаблона. Как правило, это делается для того, чтобы подчеркнуть или изменить приоритет логических шаблонов, как показано в следующем примере:

C#Копировать

if (input is not (float or double))

{

return;

}

Возможность использовать сопоставление шаблонов впервые появилась в C# 7.0. С тех пор в каждой новой основной версии C# возможности сопоставления шаблонов расширяются. Сопоставление шаблонов поддерживают следующие выражения и операторы C#:

Выражение is

Оператор switch

Выражение switch (появилось в C# 8.0)

В этих конструкциях можно сравнить входное выражение с любым из следующих шаблонов:

Шаблон объявления: для проверки типа выражения в среде выполнения и, в случае равенства, присвоения объявленной переменной результата этого выражения. Впервые появился в C# 7.0.

Шаблон типа: для проверки типа выражения в среде выполнения. Впервые появился в C# 9.0.

Шаблон константы: для проверки того, равен ли результат выражения указанной константе. Впервые появился в C# 7.0.

Реляционные шаблоны: для сравнения результата выражения с заданной константой. Впервые появился в C# 9.0.

Логические шаблоны: для проверки того, соответствует ли выражение логической комбинации шаблонов. Впервые появился в C# 9.0.

Шаблон свойства: для проверки того, соответствуют ли свойства или поля выражения вложенным шаблонам. Впервые появился в C# 8.0.

Позиционный шаблон: для деконструкции результата выражения и проверки того, соответствуют ли результирующие значения вложенным шаблонам. Впервые появился в C# 8.0.

Шаблон var: для сравнения любых выражений и присваивания результата сравнения объявленной переменной. Впервые появился в C# 7.0.

Шаблон пустой переменной: для сравнения любых выражений. Впервые появился в C# 8.0.

Логические шаблоны, шаблоны свойств и позиционные шаблоны являются рекурсивными шаблонами. То есть, они могут содержать вложенные шаблоны.

Пример использования этих шаблонов для создания управляемого данными алгоритма можно посмотреть в разделе Учебник: использование сопоставления шаблонов для создания управляемых типами и управляемых данными алгоритмов.

Шаблоны объявления и шаблоны типов

Шаблоны объявления и шаблоны типов используются для проверки того, совместим ли с указанным типом тип определенного выражения в среде выполнения. С помощью шаблона объявления можно также объявить новую локальную переменную. Если шаблон объявления соответствует выражению, этой переменной присваивается результат преобразованного выражения, как показано в следующем примере:

C#Копировать

object greeting = "Hello, World!";

if (greeting is string message)

{

Console.WriteLine(message.ToLower()); // output: hello, world!

}

Начиная с C# 7.0, шаблон объявления с типом T соответствует выражению в том случае, если результат выражения имеет значение, отличное от NULL, и выполняется любое из следующих условий:

Тип результата выражения в среде выполнения — T.

Тип результата выражения в среде выполнения является производным от типа T, реализует интерфейс T или существует другое неявное преобразование ссылок из него в T. В следующем примере показаны два случая, когда данное условие истинно:

C#Копировать

var numbers = new int[] { 10, 20, 30 };

Console.WriteLine(GetSourceLabel(numbers)); // output: 1

var letters = new List<char> { 'a', 'b', 'c', 'd' };

Console.WriteLine(GetSourceLabel(letters)); // output: 2

static int GetSourceLabel<T>(IEnumerable<T> source) => source switch

{

Array array => 1,

ICollection<T> collection => 2,

\_ => 3,

};

В предыдущем примере при первом вызове метода GetSourceLabel первый шаблон соответствует значению аргумента, так как тип этого аргумента (int[]) в среде выполнения является производным от типа Array. При втором вызове метода GetSourceLabel тип аргумента в среде выполнения (List<T>) не является производным от типа Array, но реализует интерфейс ICollection<T>.

Тип результата выражения в среде выполнения является типом, допускающим значение NULL, и имеющим базовый тип T.

Существует упаковка-преобразование или распаковка-преобразование из типа результата выражения в среде выполнения в тип T.

В следующем примере показаны два последних условия:

C#Копировать

int? xNullable = 7;

int y = 23;

object yBoxed = y;

if (xNullable is int a && yBoxed is int b)

{

Console.WriteLine(a + b); // output: 30

}

Если требуется проверить только тип выражения, можно вместо имени переменной использовать пустую переменную \_, как показано в следующем примере:

C#Копировать

public abstract class Vehicle {}

public class Car : Vehicle {}

public class Truck : Vehicle {}

public static class TollCalculator

{

public static decimal CalculateToll(this Vehicle vehicle) => vehicle switch

{

Car \_ => 2.00m,

Truck \_ => 7.50m,

null => throw new ArgumentNullException(nameof(vehicle)),

\_ => throw new ArgumentException("Unknown type of a vehicle", nameof(vehicle)),

};

}

Начиная с C# 9.0, для этой цели можно использовать шаблон типа, как показано в следующем примере:

C#Копировать

public static decimal CalculateToll(this Vehicle vehicle) => vehicle switch

{

Car => 2.00m,

Truck => 7.50m,

null => throw new ArgumentNullException(nameof(vehicle)),

\_ => throw new ArgumentException("Unknown type of a vehicle", nameof(vehicle)),

};

Как и шаблон объявления, шаблон типа соответствует выражению, если результат выражения не равен NULL, а его тип в среде выполнения удовлетворяет любому из указанных выше условий.

Дополнительные сведения см. в разделах Шаблон объявления и Шаблон типа в примечаниях к предлагаемой функции.

Шаблон константы

Начиная с C# 7.0, вы можете использовать шаблон константы для проверки того, равен ли результат выражения заданной константе, как показано в следующем примере:

C#Копировать

public static decimal GetGroupTicketPrice(int visitorCount) => visitorCount switch

{

1 => 12.0m,

2 => 20.0m,

3 => 27.0m,

4 => 32.0m,

0 => 0.0m,

\_ => throw new ArgumentException($"Not supported number of visitors: {visitorCount}", nameof(visitorCount)),

};

В шаблоне константы можно использовать любое константное выражение, например:

целочисленный литерал или литерал с плавающей точкой;

символьный или строковый литерал;

логическое значение true или false;

значение типа enum;

имя объявленного поля с модификатором const или локального элемента.

null

Используйте шаблон константы для проверки на null, как показано в следующем примере:

C#Копировать

if (input is null)

{

return;

}

Компилятор гарантирует, что при вычислении выражения x is null не будет вызван перегруженный пользователем оператор равенства ==.

Начиная с C# 9.0, вы можете использовать шаблон константы null с отрицанием для проверки неравенства значению NULL, как показано в следующем примере:

C#Копировать

if (input is not null)

{

// ...

}

Дополнительные сведения см. в разделе Шаблон константы в примечании к предлагаемой функции.

Реляционные шаблоны

Начиная с C# 9.0, вы можете использовать реляционный шаблон для сравнения результата выражения с константой, как показано в следующем примере:

C#Копировать

Console.WriteLine(Classify(13)); // output: Too high

Console.WriteLine(Classify(double.NaN)); // output: Unknown

Console.WriteLine(Classify(2.4)); // output: Acceptable

static string Classify(double measurement) => measurement switch

{

< -4.0 => "Too low",

> 10.0 => "Too high",

double.NaN => "Unknown",

\_ => "Acceptable",

};

В реляционном шаблоне можно использовать любые операторы отношений — <, >, <=, >=. Правая часть реляционного шаблона должна быть константным выражением. Константное выражение может быть целым числом, числом с плавающей точкой, символом или иметь тип enum.

Чтобы проверить, находится ли результат выражения в определенном диапазоне, сопоставьте его с шаблоном конъюнкции (and), как показано в следующем примере:

C#Копировать

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 3, 14))); // output: spring

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 7, 19))); // output: summer

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 2, 17))); // output: winter

static string GetCalendarSeason(DateTime date) => date.Month switch

{

>= 3 and < 6 => "spring",

>= 6 and < 9 => "summer",

>= 9 and < 12 => "autumn",

12 or (>= 1 and < 3) => "winter",

\_ => throw new ArgumentOutOfRangeException(nameof(date), $"Date with unexpected month: {date.Month}."),

};

Если результат выражения — null или его не удается преобразовать в тип константы с помощью преобразования, допускающего значение NULL, или распаковки-преобразования, то реляционный шаблон не соответствует выражению.

Дополнительные сведения см. в разделе Реляционные шаблоны в примечании к предлагаемой функции.

Логические шаблоны

Начиная с C# 9.0, вы можете использовать блоки объединения not, and и or для создания следующих логических шаблонов:

Шаблон отрицания not, который соответствует выражению, если шаблон с отрицанием не соответствует выражению. В следующем примере показано, как можно инвертировать шаблон константы null, чтобы проверить, имеет ли выражение значение, отличное от NULL:

C#Копировать

if (input is not null)

{

// ...

}

Шаблон конъюнкции and, который соответствует выражению, если оба шаблона соответствуют этому выражению. В следующем примере показано, как можно объединить реляционные шаблоны, чтобы проверить, находится ли значение в определенном диапазоне:

C#Копировать

Console.WriteLine(Classify(13)); // output: High

Console.WriteLine(Classify(-100)); // output: Too low

Console.WriteLine(Classify(5.7)); // output: Acceptable

static string Classify(double measurement) => measurement switch

{

< -40.0 => "Too low",

>= -40.0 and < 0 => "Low",

>= 0 and < 10.0 => "Acceptable",

>= 10.0 and < 20.0 => "High",

>= 20.0 => "Too high",

double.NaN => "Unknown",

};

Шаблон дизъюнкции or, который соответствует выражению, если любой из шаблонов соответствует данному выражению, как показано в следующем примере:

C#Копировать

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 1, 19))); // output: winter

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 10, 9))); // output: autumn

Console.WriteLine(GetCalendarSeason(new DateTime(2021, 5, 11))); // output: spring

static string GetCalendarSeason(DateTime date) => date.Month switch

{

3 or 4 or 5 => "spring",

6 or 7 or 8 => "summer",

9 or 10 or 11 => "autumn",

12 or 1 or 2 => "winter",

\_ => throw new ArgumentOutOfRangeException(nameof(date), $"Date with unexpected month: {date.Month}."),

};

Как показано в предыдущем примере, блоки объединения в шаблоне можно использовать многократно.

Блок объединения шаблонов and имеет более высокий приоритет, чем or. Чтобы явно задать приоритет, используйте круглые скобки, как показано в следующем примере:

C#Копировать

static bool IsLetter(char c) => c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z');

**Примечание**

Порядок проверки шаблонов не определен. Во время выполнения можно сначала проверить правые вложенные шаблоны шаблонов or и and.

Дополнительные сведения см. в разделе Блоки объединения шаблонов в примечании к предлагаемой функции.

Шаблон свойства

Начиная с C# 8.0, вы можете использовать шаблон свойства для сопоставления свойств или полей выражения с вложенными шаблонами, как показано в следующем примере:

C#Копировать

static bool IsConferenceDay(DateTime date) => date is { Year: 2020, Month: 5, Day: 19 or 20 or 21 };

Шаблон свойства соответствует выражению, если результат выражения не равен NULL, а каждый вложенный шаблон соответствует соответствующему свойству или полю результата выражения.

Вы также можете добавить проверку типа среды выполнения и объявление переменной в шаблон свойства, как показано в следующем примере:

C#Копировать

Console.WriteLine(TakeFive("Hello, world!")); // output: Hello

Console.WriteLine(TakeFive("Hi!")); // output: Hi!

Console.WriteLine(TakeFive(new[] { '1', '2', '3', '4', '5', '6', '7' })); // output: 12345

Console.WriteLine(TakeFive(new[] { 'a', 'b', 'c' })); // output: abc

static string TakeFive(object input) => input switch

{

string { Length: >= 5 } s => s.Substring(0, 5),

string s => s,

ICollection<char> { Count: >= 5 } symbols => new string(symbols.Take(5).ToArray()),

ICollection<char> symbols => new string(symbols.ToArray()),

null => throw new ArgumentNullException(nameof(input)),

\_ => throw new ArgumentException("Not supported input type."),

};

Шаблон свойства является рекурсивным шаблоном. Это значит, что любой шаблон можно использовать как вложенный шаблон. Используйте шаблон свойства для сопоставления элементов данных с вложенными шаблонами, как показано в следующем примере:

C#Копировать

public record Point(int X, int Y);

public record Segment(Point Start, Point End);

static bool IsAnyEndOnXAxis(Segment segment) =>

segment is { Start: { Y: 0 } } or { End: { Y: 0 } };

В предыдущем примере используются две возможности, доступные в C# 9.0 и более поздних версиях языка: блок объединения шаблонов orи типы записей.

Начиная с C# 10 можно ссылаться на вложенные свойства или поля в шаблоне свойства. Например, можно выполнить рефакторинг метода из предыдущего примера в следующий эквивалентный код:

C#Копировать

static bool IsAnyEndOnXAxis(Segment segment) =>

segment is { Start.Y: 0 } or { End.Y: 0 };

Дополнительные сведения см. в разделе Шаблон свойства в примечании к предлагаемой функции и примечание к предлагаемой функции Расширенные шаблоны свойств.

Позиционный шаблон

Начиная с C# 8.0, вы можете использовать позиционный шаблон для деконструкции результата выражения и сравнения результирующих значений с соответствующими вложенными шаблонами, как показано в следующем примере:

C#Копировать

public readonly struct Point

{

public int X { get; }

public int Y { get; }

public Point(int x, int y) => (X, Y) = (x, y);

public void Deconstruct(out int x, out int y) => (x, y) = (X, Y);

}

static string Classify(Point point) => point switch

{

(0, 0) => "Origin",

(1, 0) => "positive X basis end",

(0, 1) => "positive Y basis end",

\_ => "Just a point",

};

В предыдущем примере тип выражения содержит метод Deconstruct, используемый для деконструкции результата выражения. Можно также сопоставлять выражения кортежных типов с позиционными шаблонами. Таким образом можно сопоставить несколько входных значений с различными шаблонами, как показано в следующем примере:

C#Копировать

static decimal GetGroupTicketPriceDiscount(int groupSize, DateTime visitDate)

=> (groupSize, visitDate.DayOfWeek) switch

{

(<= 0, \_) => throw new ArgumentException("Group size must be positive."),

(\_, DayOfWeek.Saturday or DayOfWeek.Sunday) => 0.0m,

(>= 5 and < 10, DayOfWeek.Monday) => 20.0m,

(>= 10, DayOfWeek.Monday) => 30.0m,

(>= 5 and < 10, \_) => 12.0m,

(>= 10, \_) => 15.0m,

\_ => 0.0m,

};

В предыдущем примере используются реляционные и логические шаблоны, доступные в C# 9.0 и более поздних версиях языка.

В позиционном шаблоне можно использовать имена элементов кортежа и параметры метода Deconstruct, как показано в следующем примере:

C#Копировать

var numbers = new List<int> { 1, 2, 3 };

if (SumAndCount(numbers) is (Sum: var sum, Count: > 0))

{

Console.WriteLine($"Sum of [{string.Join(" ", numbers)}] is {sum}"); // output: Sum of [1 2 3] is 6

}

static (double Sum, int Count) SumAndCount(IEnumerable<int> numbers)

{

int sum = 0;

int count = 0;

foreach (int number in numbers)

{

sum += number;

count++;

}

return (sum, count);

}

Можно также расширить позиционный шаблон одним из следующих способов:

Добавьте проверку типа среды выполнения и объявление переменной, как показано в следующем примере:

C#Копировать

public record Point2D(int X, int Y);

public record Point3D(int X, int Y, int Z);

static string PrintIfAllCoordinatesArePositive(object point) => point switch

{

Point2D (> 0, > 0) p => p.ToString(),

Point3D (> 0, > 0, > 0) p => p.ToString(),

\_ => string.Empty,

};

В предыдущем примере используются позиционные записи, которые неявно обеспечивают выполнение метода Deconstruct.

Используйте шаблон свойства в позиционном шаблоне, как показано в следующем примере:

C#Копировать

public record WeightedPoint(int X, int Y)

{

public double Weight { get; set; }

}

static bool IsInDomain(WeightedPoint point) => point is (>= 0, >= 0) { Weight: >= 0.0 };

Можно объединить два предыдущих варианта, как показано в следующем примере:

C#Копировать

if (input is WeightedPoint (> 0, > 0) { Weight: > 0.0 } p)

{

// ..

}

Позиционный шаблон является рекурсивным шаблоном. Это значит, что любой шаблон можно использовать как вложенный шаблон.

Дополнительные сведения см. в разделе Позиционный шаблон в примечании к предлагаемой функции.

Шаблон var

Начиная с C# 7.0, шаблон var можно использовать для сопоставления любого выражения, включая null, и присвоения результата сопоставления новой локальной переменной, как показано в следующем примере:

C#Копировать

static bool IsAcceptable(int id, int absLimit) =>

SimulateDataFetch(id) is var results

&& results.Min() >= -absLimit

&& results.Max() <= absLimit;

static int[] SimulateDataFetch(int id)

{

var rand = new Random();

return Enumerable

.Range(start: 0, count: 5)

.Select(s => rand.Next(minValue: -10, maxValue: 11))

.ToArray();

}

Шаблон var полезно использовать, если в логическом выражении вам требуется временная переменная для хранения результатов промежуточных вычислений. Шаблон var также можно использовать, если требуется реализовать дополнительные проверки в условиях регистра when выражения или оператора switch, как показано в следующем примере:

C#Копировать

public record Point(int X, int Y);

static Point Transform(Point point) => point switch

{

var (x, y) when x < y => new Point(-x, y),

var (x, y) when x > y => new Point(x, -y),

var (x, y) => new Point(x, y),

};

static void TestTransform()

{

Console.WriteLine(Transform(new Point(1, 2))); // output: Point { X = -1, Y = 2 }

Console.WriteLine(Transform(new Point(5, 2))); // output: Point { X = 5, Y = -2 }

}

В предыдущем примере шаблон var (x, y) эквивалентен позиционному шаблону (var x, var y).

В шаблоне var тип объявленной переменной равен установленному во время компиляции типу выражения, сопоставляемого с данным шаблоном.

Дополнительные сведения см. в разделе Шаблон var в примечании к предлагаемой функции.

Шаблон пустой переменной

Начиная с C# 8.0, шаблон пустой переменной \_ используется для сопоставления любых выражений, включая null, как показано в следующем примере:

C#Копировать

Console.WriteLine(GetDiscountInPercent(DayOfWeek.Friday)); // output: 5.0

Console.WriteLine(GetDiscountInPercent(null)); // output: 0.0

Console.WriteLine(GetDiscountInPercent((DayOfWeek)10)); // output: 0.0

static decimal GetDiscountInPercent(DayOfWeek? dayOfWeek) => dayOfWeek switch

{

DayOfWeek.Monday => 0.5m,

DayOfWeek.Tuesday => 12.5m,

DayOfWeek.Wednesday => 7.5m,

DayOfWeek.Thursday => 12.5m,

DayOfWeek.Friday => 5.0m,

DayOfWeek.Saturday => 2.5m,

DayOfWeek.Sunday => 2.0m,

\_ => 0.0m,

};

В предыдущем примере шаблон пустой переменной используется для обработки значения null и любых целочисленных значений, которые не соответствуют имеющимся членам перечисления DayOfWeek. Благодаря этому гарантируется, что выражение switch в приведенном примере сможет обработать все возможные входные значения. Если в выражении switch не используется шаблон пустой переменной и при этом ни один из шаблонов выражения не соответствует входным данным, среда выполнения генерирует исключение. Если выражение switch не обрабатывает все возможные входные значения, компилятор генерирует предупреждение.

Шаблон пустой переменной не может быть шаблоном в выражении is или операторе switch. В этих случаях для сопоставления выражений используйте шаблон var с пустой переменной: var \_.

Дополнительные сведения см. в разделе Шаблон пустой переменной в примечании к предлагаемой функции.

Шаблон в круглых скобках

Начиная с C# 9.0, вы можете использовать круглые скобки вокруг любого шаблона. Как правило, это делается для того, чтобы подчеркнуть или изменить приоритет логических шаблонов, как показано в следующем примере:

C#Копировать

if (input is not (float or double))

{

return;

}

9 Дайте определение понятию «Пространство имен».Перечислите пространства имен, используемые в Вашей программе.

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.). Пространства имен используются для организации кода в виде логических групп и с целью избежания конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек. Все идентификаторы в пределах пространства имен доступны друг другу без уточнения.

System

System.Console