МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**ОСНОВЫ РАБОТЫ С F#** — **ЕДИНИЦЫ ИЗМЕРЕНИЯ, КОРТЕЖ, ЗАПИСЬ. ПРОСТРАНСТВО ИМЕН. МОДУЛЬ. ИНТЕРФЕЙС. АЛГЕБРАИЧЕСКИЙ ТИП.**

ЛАБОРАТОРНАЯ РАБОТА №4

(Вариант №5)

студента 4 курса 451 группы

направления 09.03.04 — Программная инженерия

факультета КНиИТ

Голикова Артема Олеговича

Проверил

к. пед. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_ В. А. Векслер

Саратов 2020

**Часть 1-я**

**«Единицы измерения, кортежи и записи»**

**Цель работы**: ознакомиться с понятием единиц измерения, кортежа и записи, на основе нескольких тренировочных программ научиться их использовать в программировании на F#.

**Краткие теоритические сведения.**

***Единицы измерения***

Как правило, программы работают не просто с переменными типа int или float. Например, метод System.String.Length возвращает не просто значение типа int, а количество символов в строке, и складывать его с другим значением того же типа int, содержащим количество секунд, прошедших с 1-го января 1970-го года, бессмысленно. Язык программирования F# позволяет программисту связать с любым значением числового типа произвольную единицу измерения. В дальнейшем, во время компиляции, будет выполняться проверка корректности приложения, на основе анализа размерностей.

Важной особенностью научных, инженерных и бизнес-расчетов являются единицы, такие как килограммы, футы, метры, ньютоны, доллары, евро и т. Д. Обычно компьютерные программы явно не учитывают единицы, и разработчик и / или пользователь должны убедиться, что вычисления правильны.

F # имеет уникальную функцию «единиц измерения», которая позволяет программисту назначать единицы для любой переменной. Затем компилятор может проверить эти блоки во время компиляции, чтобы отмечать любые ошибки или несоответствия, прежде чем код будет работать. Эта функция может значительно сократить время разработки, отладки и развертывания кода за счет обеспечения согласованности и надежности кода.

Синтаксис:

**[<Measure>] type unit-name [ = measure ]**

> [<Measure>] type m

[<Measure>] type s;;

После этого типы s и m можно использовать в качестве единиц измерения, для чего их следует указать в угловых скобках после соответствующего значения или типа:

> let a = 10<m>;;

val a : int<m> = 10

> let b = 2<s>;;

val b : int<s> = 2

Значения a и b имеют одинаковые типы, но разные размерности, что означает, что их нельзя вычитать, складывать и сравнивать. Значения разных размерностей, однако, можно умножать и делить, результат операции в таком случае будет иметь составную размерность.

> let c = a + b;;

Попытка сложить метры с секундами приводит к ошибке *«error FS0001: The unit of measure ’s’ does not match the unit of measure ’m’»*.

> let v = a/b;;

val v : int<m/s> = 5

Значение v имеет составную размерность m/s. В предыдущем примере, единицы измерения m и s являются независимыми, то есть не выражаются друг через друга, Однако F# позволяет определять зависимости между разными единицами измерения:

> [<Measure>] type Hz = s ^ -1;;

или просто:

> [<Measure>] type Hz = 1/s;;

Значения, имеющие размерность Hz, можно будет использовать в качестве значений с размерностью 1/s.

Чтобы избежать ненужных преобразований, можно не указывать единицу измерения явно. Для этого вместо имени размерности следует использовать символ подчёркивания, в результате чего механизм вывода типов определит используемую размерность автоматически, иначе она будет обобщённой.

> let sqr (a : int<\_>) = a\*a;;

val sqr : int<'u> -> int<'u ^ 2>

> sqr a;;

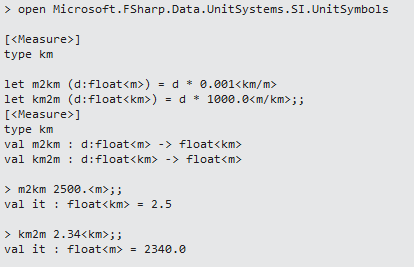
val it : int<s ^ 2> = 4

В данном примере функция sqr принимает параметр типа int, имеющий любую размерность, либо не имеющий её вовсе.

Следует также отметить, что единицы измерения не вписываются в рамки общей системы типов .NET Framework, поэтому информация о них сохраняется в метаданных и может быть использована только из другой F#-сборки.

Преобразование (переход из одной единицы измерения в другую)

В примере описано преобразование из километров в метры и обратно.



**Кортеж**

Кортеж (Tuple) — это группировка неименованных, но упорядоченных значений, возможно, различных типов.

Примерами кортежей являются пары, тройки и т. п. одного и того же или разных типов. Некоторые примеры показаны в следующем коде.

// Tuple of two integers.

( 1, 2 )

// Triple of strings.

( "one", "two", "three" )

// Tuple of unknown types.

( a, b )

// Tuple that has mixed types.

( "one", "1", "2.0" )

// Tuple of integer expressions.

( a + 1, b + 1)

Чтобы создать экземпляр кортежа, достаточно указать список значений, разделенных запятыми и (необязательно) заключить его в круглые скобки. Тип кортежа обозначается с помощью списка типов его элементов, разделенных символами \*. В следующем примере dinner – это экземпляр кортежа, а string \* string – это тип данного кортежа:

> let dinner = (“green eggs”, “ham”);;

val dinner : string \* string = (“green eggs”, “ham”)

Кортежи могут содержать любое количество элементов любых типов. Фактически можно даже создать кортеж, который будет содержать другие кортежи!

> ((1,1),(2.,2.));;

val it : (int \* int) \* (float \* float) = ((1, 1), (2.0, 2.0))

Следующий фрагмент определяет два кортежа. Первый, с именем zeros, представляет собой кортеж, содержащий различные представления нулевых значений. Второй, nested, содержит вложенные кортежи. Этот кортеж содержит три элемента, из которых второй и третий сами являются кортежами:

> let zeros = (0, 0L, 0I, 0.0);;

val zeros : int \* int64 \* bigint \* float = (0, 0L, 0I, 0.0)

> let nested = (1, (2.0, 3M), (4L, “5”, ‘6’));;

val nested : int \* (float \* decimal) \* (int64 \* string \* char)

Для извлечения значений из кортежа, содержащего два элемента, можно использовать функции fst и snd. Функция fst возвращает первый элемент кортежа, а функция snd – второй:

> let nameTuple = (“John”, “Smith”);;

val nameTuple : string \* string = (“John”, “Smith”)

> fst nameTuple;;

val it : string = “John”

> snd nameTuple;;

val it : string = “Smith”

Альтернативный способ извлечения элементов кортежей заключается в использовании оператора связывания let. Если сразу за оператором let указать несколько идентификаторов, разделенных запятыми, с этими именами будут связаны значения элементов кортежа. В следующем примере создается кортеж с именем snacks, а затем из него извлекаются значения элементов и связываются с тремя новыми идентификаторами x, y и z:

> let snacks = (“Soda”, “Cookies”, “Candy”);;

val snacks : string \* string \* string = (“Soda”, “Cookies”, “Candy”)

> let x, y, z = snacks;;

val z : string = “Soda”

val y : string = “Cookies”

val x : string = “Candy”

> y, z;;

val it : string \* string = (“Cookies”, “Candy”)

Вы получите ошибку компиляции, если попытаетесь извлечь из кортежа слишком много или слишком мало значений:

> let x, y = snacks;;

let x, y = snacks;;

-----------^^^^^^

stdin(8,12): error FS0001: Type mismatch. Expecting a

string \* string

but given a

string \* string \* string.

The tuples have differing lengths of 2 and 3.

(Несоответствие  типов.  Требуется  string  \*  string,  но  получен  string  \*

string \* string. Кортежи имеют разную длину – 2 и 3.)

Кортежи можно передавать функциям в виде параметров, как и любые другие значения.

В следующем примере функция tupledAdd принимает два параметра, x и y, в виде кортежа. Обратите внимание на различия в сигнатурах функций add и tupledAdd:

> let add x y = x + y;;

val add : int -> int -> int

> let tupledAdd(x, y) = x + y;;

val tupledAdd : int \* int -> int

> add 3 7;;

val it : int = 10

> tupledAdd(3, 7);;

val it : int = 10

Для доступа к элементам кортежа и присвоения им имен можно использовать подбор шаблона, как показано в следующем коде.

let print tuple1 =

match tuple1 with

| (a, b) -> printfn "Pair %A %A" a b;;

print (1,1);;

>

Pair 1 1

val it : unit = ()

Шаблоны кортежей можно использовать в привязках let следующим образом.

let (a, b) = (1, 2)

Здесь привязываются одновременно значения a и b. Если требуется только один элемент кортежа, можно использовать подстановочный знак (символ подчеркивания) во избежание создания нового имени для ненужной переменной.

let (a, \_) = (1, 2)

Функции fst и snd возвращают первый и второй элементы кортежа соответственно.

let c = fst (1, 2)

let d = snd (1, 2)

Встроенной функции, возвращающей третий элемент тройки, нет, однако ее легко написать следующим образом.

let third (\_, \_, c) = c

В общем случае для доступа к отдельным элементам кортежа лучше использовать подбор шаблона.

*Использование кортежей*

Кортежи — это удобный способ возвращения из функции сразу нескольких значений, как показано в следующем примере. В этом примере выполняется целочисленное деление и возвращается округленный результат операции в качестве первого члена кортежной пары и остаток в качестве второго члена пары.

let divRem a b =

let x = a / b

let y = a % b

(x, y)

Кортежи также можно использовать в качестве аргументов функций, когда требуется избежать неявного каррирования аргументов функции, что подразумевается обычным синтаксисом функций.

let sumNoCurry (a, b) = a + b

Обычный синтаксис определения функции let sum a b = a + b позволяет определить функцию, являющуюся частичным применением первого аргумента функции, как показано в следующем коде.

let addTen = sum 10

let result = addTen 95

// Result is 105.

Использование кортежа в качестве параметра предотвращает каррирование.

*Имена кортежных типов*

При записи имени типа, который является кортежем, для разделения элементов используется символ \*. В случае кортежа, состоящего из int, float и string, такого как (10, 10.0, "ten"), тип будет записан следующим образом.

int \* float \* string

*Записи (record)* в F# аналогичны кортежам, с той разницей, что в них каждое поле имеет название. Определение записи заключается в фигурные скобки и разделяется точкой с запятой.

type org = { boss : string; tops :int };;

let org1 = { boss = "Bill Gates"; tops = 2};;

Доступ к полям записи осуществляется, как обычно, через точку.

org1.boss;;

> val it : string = "Bill Gates"

> // Определение типа записи

type PersonRec = { First : string; Last : string; Age : int};;

type PersonRec =

{First: string;

Last: string;

Age: int;}

> // Конструирование экземпляра записи

let steve = { First = “Steve”; Last = “Holt”; Age = 17 };;

val steve : PersonRec = {First = “Steve”;

Last = “Holt”;

Age = 17;}

> // Использование ‘.field’

// для доступа к полям записи

printfn “%s is %d years old” steve.First steve.Age;;

Steve is 17 years old

Записи легко могут клонироваться с помощью ключевого слова with:

type Car =

{

Make : string

Model : string

Year : int

}

let thisYear’s = { Make = “FSharp”; Model = “Luxury Sedan”; Year = 2010 }

let nextYear’s = { thisYear’s with Year = 2011 }

Что эквивалентно следующему:

let nextYear’s =

{

Make = thisYear’s.Make

Model = thisYear’s.Model

Year = 2011

}

Изменение записи:

type Car =

{

Make : string

Model : string

Year : int

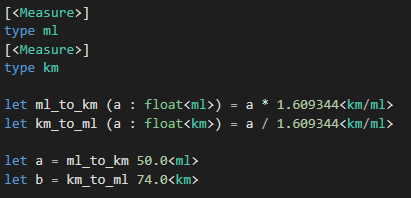
}

let mutable thisYear = { Make = "FSharp"; Model = "Luxury Sedan"; Year = 2010 }

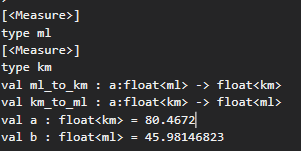
thisYear <- { thisYear with Year = 2011 }

**Ход работы:**

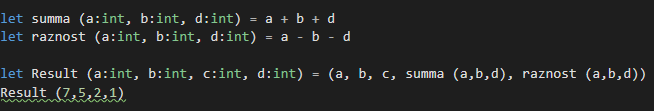
**Задание №1**. Преобразования единиц измерения (в обе стороны). Миля – Километр



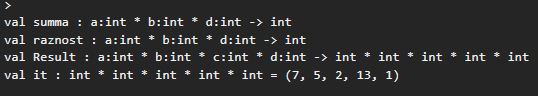
Результат выполнения:



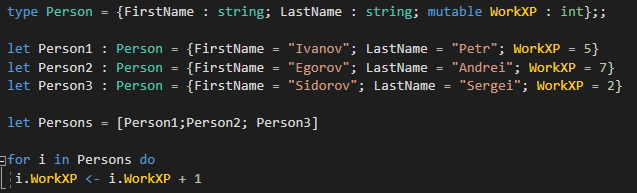
**Задание №2**. Получите первую, вторую и четвертую цифру из кортежа. Найдите их сумму и разность. Результат выведите в кортеже в виде (первая цифра, вторая цифра, третья цифра, сумма, разность).



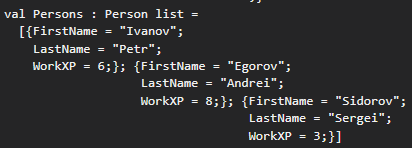
Результат выполнения:



**Задание №3.** Создайте три записи о сотрудниках (Фамилия, Имя, Стаж). Увеличьте всем стаж на 1 год.



Результат выполнения:



**Контрольные вопросы**

1. Определите понятие «единица измерения».

F # имеет уникальную функцию «единиц измерения», которая позволяет программисту назначать единицы для любой переменной. Затем компилятор может проверить эти блоки во время компиляции, чтобы отмечать любые ошибки или несоответствия, прежде чем код будет работать.

1. Дайте определение кортежу.

Кортеж (Tuple) — это группировка неименованных, но упорядоченных значений, возможно, различных типов.

1. Чем удобно использование кортежей?

Кортежи — это удобный способ возвращения из функции сразу нескольких значений, а так же удобный способ передачи значений в функцию.

1. С помощью каких функций возвращаются элементы кортежей?

Функции fst и snd возвращают первый и второй элементы кортежа соответственно. В общем случае для доступа к отдельным элементам кортежа лучше использовать подбор шаблона.

1. Каким образом задаётся кортеж?

Чтобы создать экземпляр кортежа, достаточно указать список значений, разделенных запятыми и (необязательно) заключить его в круглые скобки. Тип кортежа обозначается с помощью списка типов его элементов, разделенных символами \*.

1. Каким образом определяются имена кортежных типов?

При записи имени типа, который является кортежем, для разделения элементов используется символ \*.

1. Как можно осуществить доступ к элементам кортежа?

Функции fst и snd возвращают первый и второй элементы кортежа соответственно. В общем случае для доступа к отдельным элементам кортежа лучше использовать подбор шаблона.

1. Каких типов являются элементы кортежей?

Кортежи могут содержать любое количество элементов любых типов. Фактически можно даже создать кортеж, который будет содержать другие кортежи.

1. Что именно было создано: «val it : (int \* int) \* (float \* float)»

Кортеж из двух кортежей, где первый кортеж – пара целочисленных значений, второй – пара значений типа float.

1. Объясните код

[<Measure>]

type C

[<Measure>]

type F

let centToFahr t = 1.8<F/C> \* t + 32.0<F>

Создаются типы «Градус по Цельсию» и «Градус по Фаренгейту». Функция переводит значения из первого типа во второй.

1. Создайте программу для данного фрагмента кода

let f (m:float<kg>) (a:float<m/s^2>) = m \* a

[<Measure>]

type kg

[<Measure>]

type m

[<Measure>]

type s

[<Measure>]

type n = kg \* m / s ^ 2;;

let f (m:float<kg>) (a:float<n>) = m \* a

**Часть 2-я**

# Пространство имен. Модули. [Интеграция C# и F#](https://www.outcoldman.com/ru/archive/2009/02/28/интеграция-c-и-f/). Классы: Интерфейсы.

1. **Цель работы**: ознакомиться с понятием интерфейса, абстрактного класса на основе тренировочного задания, научиться их использовать в программировании на F#. Научиться интегрировать классы F# в приложения на C#.
2. **Краткие теоритические сведения.**

С помощью пространства имен вы можете упорядочить код по областям соответствующей функциональности, подключив имя к группированию элементов программы.

Пространства имен не может непосредственно содержат значения и функции. Вместо этого значения и функции должны быть включены в модули и модули, включаются в пространствах имен. Пространства имен может содержать типы, модули.

Пространства имен может быть объявлен явным образом с помощью ключевого слова пространства имен или неявно при объявлении модуля. Чтобы объявить пространство имен явно, используйте ключевое слово namespace, за которым следует имя пространства имен. Следующий пример показывает файл кода, объявляющий пространство имен мини-приложения с типом и модуль, включенный в этом пространстве имен.

В контексте языка F# модуль — это группирование кода F#, например значений, типов и значений функций, в программе F#. Код группирования в модулях объединяет связанный код и помогает избежать конфликтов имен в программе.

Модуль F# реализуется как общий класс языковой среды выполнения (CLR), содержит только статические члены.

**Интерфейсы**— важная часть платформы .NET. Они позволяют описать контракт, который обязуется выполнить класс, реализуя методы данного интерфейса. Если обычное наследование реализует отношение «является частным случаем» и применяется для расширения базового класса, то наследование от интерфейса реализует отношение «поддерживает» и применяется для того, чтобы определить возможные точки соприкосновения разных подсистем, которые могут ничего не знать друг о друге, кроме того, что одна из них поддерживает нужный интерфейс.

Интерфейс в F# — это просто чистый абстрактный класс.

Реализация интерфейса отличается от наследования класса, что, впрочем, логично. Нелогичным может показаться то, что нельзя вызывать перегруженные методы интерфейса — это приведет к ошибке компиляции. Чтобы вызвать метод интерфейса, нужно выполнить приведение типа с помощью оператора ‘:>’.

**Индивидуальное задание.**

Класс с интерфейсами реализуйте в отдельной библиотеке. Дополнительно реализуйте в классе интерфейс IComparer. Продемонстрируйте интеграцию класса с проектом на С#.

|  |  |  |
| --- | --- | --- |
| **№ вар.** | **Классы** | **Интерфейсы** |
| **5, 13,**  **21** | ***Посылка*** (код, отправитель, адрес получателя, вес, получить данные). | ***Перемещаемый в пространстве*** (текущие координаты, заданные координаты, перемещать к цели). |

Код на F#:

namespace Lab4Library

open System

open System.Collections.Generic

type Move =

abstract CurrentLocation : string with get, set

abstract NeedLocation : string with get

abstract IsMove : bool with get

abstract Print : string with get

type Package(code: int, name : string, sender: string, address: string, weight: float) =

let mutable \_code = code

let mutable \_name = name

let mutable \_sender = sender

let mutable \_address = address

let mutable \_weight = weight

let mutable \_curloc = "PostOffice"

member this.Code

with get() = \_code

and set(value) = \_code <- value

member this.Address

with get() = \_address

and set(value) = \_address <- value

member this.Name

with get() = \_name

and set(value) = \_name <- value

member this.Sender

with get() = \_sender

and set(value) = \_sender <- value

member this.Weight

with get() = \_weight

and set(value) = \_weight <- value

member this.CurrLoc

with get() = \_curloc

and set(value) = \_curloc <- value

interface Move with

member this.Print

with get() =

" Package Code: " + string(this.Code) + "\n Contains: " + this.Name + "\n Sender: "

+ this.Sender + "\n Address: " + this.Address + "\n Weight: " + string(this.Weight) + "\n\n"

member this.CurrentLocation

with get() = this.CurrLoc

and set(value) = this.CurrLoc <- value

member this.NeedLocation

with get() = this.Address

member this.IsMove =

if (System.String.Equals(this.CurrLoc, this.Address, System.StringComparison.CurrentCultureIgnoreCase)) then false

else true

interface IComparable<Package> with

member this.CompareTo (b) =

this.Code.CompareTo(b.Code)

Код на С#:

using System;

using Lab4Library;

namespace Lab4\_2\_GolikovAO

{

class Program

{

static void Main(string[] args)

{

// Создаем четрые объекта (посылки)

Package i1 = new Package(9123, "Запчасть", "Иванов А.С.", "Саратов", 3.54);

Package i2 = new Package(517, "Клавиатура", "Сидоров И.В.", "Челябинск", 0.921);

Package i3 = new Package(1555, "Смартфон", "Афанасьев Д.А.", "Санкт-Петербург", 1.530);

Package i4 = new Package(7712, "Книга", "Федоров К.С.", "Москва", 1.214);

Move[] r = { i1, i2, i3, i4 };

// Создаем массив посылок

var tmp = new Package[] { i1, i2, i3, i4 };

// Сортируем массив. Сортируется по коду посылки.

Array.Sort(tmp);

// Вывод информации по посылкам

foreach (Move item in tmp)

Console.WriteLine(item.Print);

// Проверяем нужно ли перемещать посылку (True - нужно)

Console.WriteLine(r[0].IsMove);

// Теперь присвоим текущему положению конечный адрес

r[0].CurrentLocation = i1.Address;

// Снова проверяем нужно ли перемещать посылку (False - не нужно)

Console.WriteLine(r[0].IsMove);

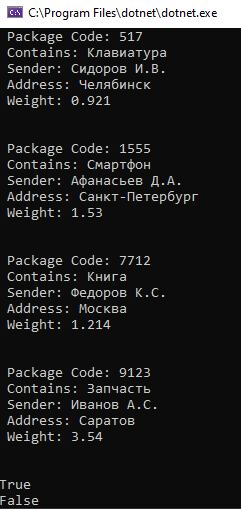
Console.ReadLine();

}

}

}

Результат работы:



**Часть 3-я**

**Алгебраический тип или "Discriminated Unions"**

1. **Цель работы**: ознакомиться с понятием алгебраического типа, на основе тренировочного задания, научиться их использовать в программировании на F#.
2. **Краткие теоритические сведения.**

Кортежи и записи являются примерами создания новых типов путем «сбора» существующих типов вместе.  Другим способом создания новых типов было «суммирование» существующих типов. Что это значит?

Ну, допустим, мы хотим определить функцию, которая работает с целыми числами или логическими значениями, возможно, чтобы преобразовать их в строки. Но мы хотим быть строгими и не принимать никакого другого типа (например, вещественные числа или строки). Следующая диаграмма представляет, то какой должна быть функция (рис. 13.5):

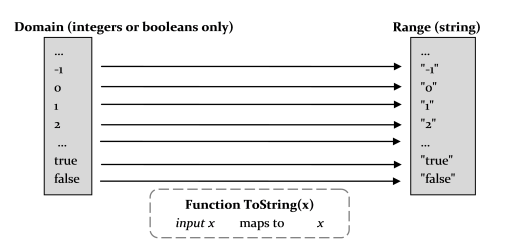


Рисунок 13.5

Как мы можем представить область этой функции?

Нам нужен тип, который представляет все возможные целые числа и все возможные булевы значения (рисунок 13.6).

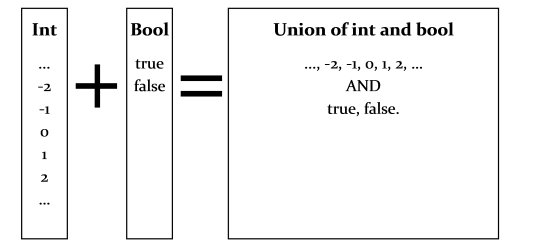


Рисунок 13.6

Другими словами, тип «сумма». В этом случае новый тип — это «сумма» целочисленного типа плюс булевский тип.

В F # тип суммы называется «дискриминационным объединенным» типом. Каждый тип компонента (называемый «случай объединения») должен быть помечен меткой (называемой «идентификатором случая» или «тегом»), для того чтобы их можно было разделить («дискриминировать»). Метки могут быть любыми идентификаторами, но должны начинаться с буквы верхнего регистра.

Вот как мы можем провести определить:

type IntOrBool =

| I of int

| B of bool

«I» и «B» - это просто произвольные метки; мы могли бы использовать любые другие ярлыки, которые были бы значимыми.

Для небольших типов мы можем поместить определение в одну строку:

type IntOrBool = I of int | B of bool

Типы компонентов могут быть любыми другими типами, включая кортежи, записи, другие типы соединений и т. Д.

type Person = {first:string; last:string} // define a record type

type IntOrBool = I of int | B of bool

type MixedType =

| Tup of int \* int // a tuple

| P of Person // use the record type defined above

| L of int list // a list of ints

| U of IntOrBool // use the union type defined above

Вы даже можете иметь типы, которые являются рекурсивными, то есть они ссылаются на себя. Обычно это определяется как древовидные структуры.

Вертикальная полоса необязательна перед первым компонентом, так что следующие определения эквивалентны:

type IntOrBool = I of int | B of bool // without initial bar

type IntOrBool = | I of int | B of bool // with initial bar

type IntOrBool =

| I of int

| B of bool // with initial bar on separate lines

Теги или метки должны начинаться с буквы верхнего регистра. Итак, следующее сообщение даст ошибку:

type IntOrBool = int of int| bool of bool

// error FS0053: Discriminated union cases

// must be uppercase identifiers

Другие именованные типы (такие как Person или  IntOrBool) должны быть предварительно определены вне типа объединения. Вы не можете определить их «внутри» и написать что-то вроде этого:

type MixedType =

| P of {first:string; last:string} // error

или же

type MixedType =

| U of (I of int | B of bool) // error

Ярлыки могут быть любым идентификатором, включая имена самого типа компонента.

open System

type IntOrBool = Int32 of Int32 | Boolean of Boolean

Этот стиль «дублирования имен» на самом деле довольно распространен.

**Индивидуальное задание.**

Создайте алгебраический тип «Фигура» (окружность с радиусом, равносторонний треугольник с длиной стороны, квадрат с длиной стороны, прямоугольник с кортежем длин сторон). Создайте методы нахождения площади, периметра, печати данных о фигуре. Продемонстрируйте использование алгебраического типа.

Код программы:

open System

// Алгебраический тип

type Figures =

| Circle of double

| Square of double

| Triangle of double

| Rectangle of double \* double

type Figure (F:Figures) =

let mutable \_F = F

member this.F

with get () = \_F

// Функция вычисления периметров

member this.Perimeter

with get() =

match \_F with

| Circle(r) -> (Math.PI \* 2.0 \* r)

| Square(a) -> (4.0 \* a)

| Triangle(a) -> (3.0 \* a)

| Rectangle(a,b) -> (2.0 \* (a + b))

// Функция вычисления площадей

member this.Area

with get() =

match \_F with

| Circle(r) -> (Math.PI \* r \* r)

| Square(a) -> (a \* a)

| Triangle(a) -> (sqrt(3.0)/4.0 \* a \* a)

| Rectangle(a,b) -> (a \* b)

// Функции вывода на экран информации о фигуре

member this.Print

with get() =

match \_F with

| Circle(r) -> printfn "Круг с радиусом %A. Периметр - %A. Площадь - %A" r this.Perimeter this.Area

| Square(a) -> printfn "Квадрат со стороной %A. Периметр - %A. Площадь - %A" a this.Perimeter this.Area

| Triangle(a) -> printfn "Правильный треугольник со стороной %A. Периметр - %A. Площадь - %A" a this.Perimeter this.Area

| Rectangle(a,b) -> printfn "Прямоугольник со сторонами %A и %A. Периметр - %A. Площадь - %A" a b this.Perimeter this.Area

[<EntryPoint>]

let main argv =

// Создаем четыре фигуры (круг, квадрат, треугольник, прямоугольник)

let f1 = new Figure(Circle(4.0))

let f2 = new Figure(Square(5.0))

let f3 = new Figure(Triangle(3.0))

let f4 = new Figure(Rectangle(4.0, 3.5))

let figures = [|f1; f2; f3; f4|]

// Выводим на экран информацию по каждой фигуре

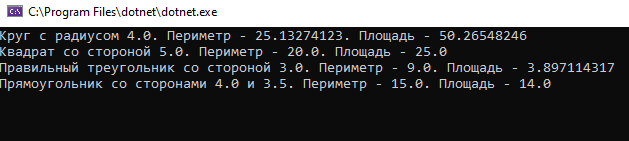
for item in figures do

item.Print

Console.ReadLine() |> ignore

0

Результат выполнения:



**Вывод:** ознакомился понятием единиц измерения, кортежа и записи, интерфейса, абстрактного класса, алгебраического типа на основе нескольких тренировочных программ. Научился их использовать в программировании на F# и интегрировать классы F# в приложения на C#.