Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №2

Создание и использование размерных типов данных

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2025

Цель работы:
Изучить размерные типы данных и приобрести навыки работы со структурными типами.

Упражнение 1. Создание перечисления.
Я создал перечисление для представления различных типов банковских счетов и использовал его для создания двух переменных, которым присвоил значения `Checking` и `Deposit`. Затем я вывел значения этих переменных при помощи функции `Console.WriteLine()`.

```
using System;

public enum AccountType { Checking, Deposit }

public class Enum
{
    public static void Main(string[] args)
    {
        AccountType goldAccount;
        AccountType platinumAccount;

        goldAccount = AccountType.Checking;
        platinumAccount = AccountType.Deposit;

        Console.WriteLine("The Customer Account Type is {0}",
goldAccount);
        Console.WriteLine("The Customer Account Type is {0}",
platinumAccount);
    }
}
```

Пример работы:

```
 labs/lab2/BankAccount  main ❯
dotnet run                                        dotnet .NET  17:53
The Customer Account Type is Checking
The Customer Account Type is Deposit
```

Упражнение 2. Создание и использование структуры.
Я создал структуру, которую можно использовать для представления банковских счетов. Для хранения номеров счетов (`long`), балансов

счетов (`decimal`) и типов счетов (`AccountType`). Затем создал переменную типа структуры, заполнил ее данными и вывел результаты на консоль.

```
using System;

public enum AccountType { Checking, Deposit }

public struct BankAccount
{
    public long accNo;
    public decimal accBal;
    public AccountType accType;
}

public class Struct
{
    public static void Main(string[] args)
    {
        BankAccount goldAccount;


        goldAccount.accType = AccountType.Checking;
        goldAccount.accBal = (decimal)3200.00;
        Console.Write("Enter account number: ");
        goldAccount.accNo = long.Parse(Console.ReadLine());

        Console.WriteLine("*** Account Summary ***");
        Console.WriteLine("Acct Number {0}",
goldAccount.accNo);
        Console.WriteLine("Acct Type {0}",
goldAccount.accType);
        Console.WriteLine("Acct Balance ${0}",
goldAccount.accBal);
    }
}
```

Пример работы:

```
 labs/lab2/StructType  main  !? 
dotnet run                                              dotnet .NET  17:55
/home/nik/oop/labs/lab2/StructType/Struct.cs(20,40): warning CS8604: Po
ssible null reference argument for parameter 's' in 'long long.Parse(st
ring s)'.
Enter account number: 1234
*** Account Summary ***
Acct Number 1234
Acct Type Checking
Acct Balance $3200
```

Упражнение 3. Реализация структуры Distance.

Создал структуру Distance, определяющую длину в английской системе мер.

```
using System;

struct Distance
{
    public int feet;
    public int inches;

    public void Print()
    {
        Console.WriteLine("{0}' - {1}''", feet, inches);
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        Distance d1, d2, d3;

        Console.Write("Input the first distance (feet): ");
        d1.feet = int.Parse(Console.ReadLine());
        Console.Write("Input the first distance (inches): ");
        d1.inches = int.Parse(Console.ReadLine());

        Console.Write("Input the second distance (feet): ");
        d2.feet = int.Parse(Console.ReadLine());
        Console.Write("Input the second distance (inches): ");
        d2.inches = int.Parse(Console.ReadLine());
```

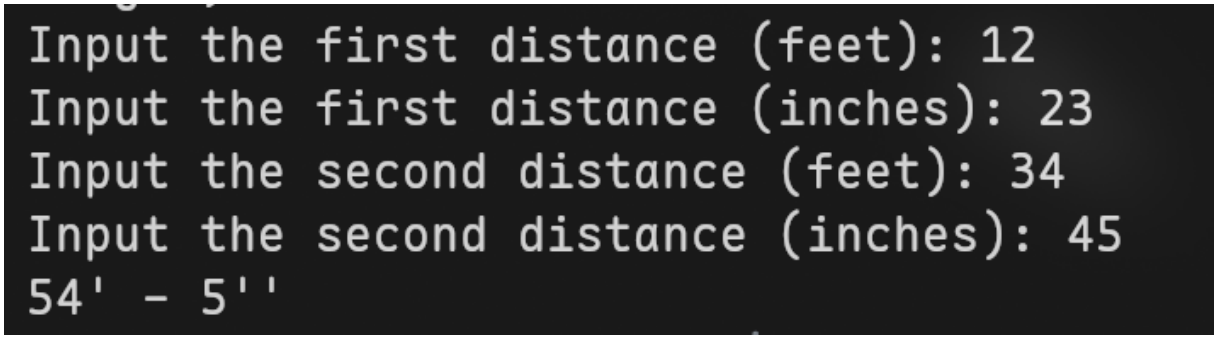
```
        d3.feet = d1.feet + d2.feet;
        d3.inches = d1.inches + d2.inches;

        d3.feet += d3.inches % 12;
        d3.inches = d3.inches / 12;

        d3.Print();

    }
}
```

Пример работы:



Выводы.
В ходе выполнения лабораторной работе я изучил размерные типы данных и приобрел навыки работы со структурными типами.

Code Review. (by zzzcode.ai)
Резюме

Код, представленный выше, демонстрирует использование перечислений, структур и базовых операций ввода-вывода в C#. Однако, он содержит несколько недостатков, которые могут повлиять на его читаемость, производительность и безопасность.

Ошибка

В коде есть несколько потенциальных ошибок, связанных с обработкой ввода. Например, если пользователь введет некорректные данные (нечисловые значения), программа вызовет исключение. Рекомендуется использовать `TryParse` для безопасного парсинга.

Стиль кода

Стиль кода в целом соответствует стандартам C#. Однако, использование имен классов Enum и Struct не является удачным, так как они могут вызвать путаницу с встроенными типами. Рекомендуется использовать более описательные имена, такие как `AccountEnum` и `BankAccountStruct`.

Структура кода

Код разбит на три отдельных класса, что хорошо для организации, но может быть улучшено. Например, можно объединить функциональность в один класс с методами, чтобы избежать дублирования кода.

Читаемость

Читаемость кода можно улучшить, добавив комментарии и более четкие имена переменных. Например, вместо `d1`, `d2`, `d3` можно использовать `firstDistance`, `secondDistance`, `resultDistance`.

Производительность

Производительность кода не вызывает серьезных проблем, однако, использование `Console.ReadLine()` в цикле может быть оптимизировано, если ввод данных будет обрабатываться в одном методе.

Масштабируемость

Код не очень масштабируемый, так как добавление новых типов счетов или новых единиц измерения потребует значительных изменений. Рекомендуется использовать более гибкие структуры данных, такие как классы или коллекции.

Безопасность

Безопасность кода может быть улучшена путем добавления обработки исключений и проверки ввода. Это поможет избежать сбоев программы при некорректном вводе данных.

Обработка ошибок

Обработка ошибок в текущем коде отсутствует. Рекомендуется добавить блоки **try-catch** для обработки возможных исключений, особенно при работе с вводом пользователя.

Заключение

Код демонстрирует базовые концепции C#, но требует доработки для повышения его надежности, читаемости и безопасности. Рекомендуется внести изменения, чтобы улучшить структуру и обработку ошибок, а также использовать более описательные имена для классов и переменных.