Министерство образования и науки Российской Федерации
ФГАОУ ВО «КФУ им. В.И. Вернадского»
физико-технический институт (структурное подразделение)

Кафедра компьютерной инженерии и моделирования

Шенгелай Всеволод Михайлович

отчет по лабораторной работе №5
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:
09.03.04 "Программная инженерия"

Оценка - 90

Симферополь, 2021

Лабораторная работа №5

Тема: Типы перечислений и структуры.

Цель работы: Научиться на практике создавать перечисления и структуры. Разобраться самостоятельно с эффективностью использования структур, недостатки и преимущества по сравнению с классами.

Описание ключевых понятий:

Enum – это тип значения, определенный набором именованных констант применяемого целочисленного типа. Чтобы определить тип перечисления, используется ключевое слово **enum** и указываются имена элементов перечисления. По умолчанию связанные значения констант элементов перечисления имеют тип **int**. Они начинаются с нуля и увеличиваются на единицу в соответствии с порядком текста определения. Мы можем явно указать любой другой целочисленный тип в качестве базового типа перечисления. Мы можетм также явно указать соответствующие значения констант.

Struct - тип структуры ппредставляет собой тип значения, который может инкапсулировать данные и связанные функции. Для определения типа структуры используется ключевое слово **struct**. Типы структуры имеют семантики значений. То есть переменная типа структуры содержит экземпляр этого типа. По умолчанию значения переменных копируются при назначении, передаче аргумента в метод и возврате результата метода. В случае переменной типа структуры копируется экземпляр типа.

Как правило, типы структуры используются для проектирования небольших ориентированных на данные типов, которые предоставляют минимум поведения или не предоставляют его вовсе. Например, платформа .NET использует типы структуры для представления числа (как целого, так и вещественного), логического значения, символа Юникода, экземпляра времени.

Nullable - в отличие от ссылочных типов переменным/параметрам значимых типов нельзя напрямую присвоить значение null. Тем не менее нередко бывает удобно,

чтобы переменная/параметр значимого типа могли принимать значение null. Например, получаем числовое значение из базы данных, которое в бд может отсутствовать. То есть, если значение в базе данных есть - получим число, если нет - то null.

Чтобы присвоить переменной или параметру значимого типа значения **null**, эти переменная/параметр значимого типа должны представлять тип **nullable**. Для этого после названия типа указывается знак вопроса **?**

```
int? val = null;
```

Перед выполнением лабораторной работы изучена следующая литература:

1. Презентация лектора курса: «Обработка исключений в C#».
2. Шилдт Герберт. Полный справочник по C#. Пер. с англ. - М.: Издательский дом "Вильямс", 2008. - 752с.: ил. (глава 12).
3. Эндрю Троелсен. Язык программирования C# и платформы .NET и .NET CORE. Часть II. Глава 4.
4. Сайт MSDN Microsoft.
5. Сайт Metanit.com

Выполнены 3 задания, описанных в методических указания к выполнению лабораторных работ.

Задание 1: В этом Задании мы определяем перечисления, которые представляют различные материалы, находящиеся под нагрузкой (нержавеющая сталь, алюминий, железобетон и титан), и поперечное сечение балок (двутавровая, прямоугольная, Z-образная и C-образная). Мы также определяем другое перечисление под названием TestResult, которое представляет результаты стресс-теста. Значения перечислений мы загружаем в элементы управления ListBox (рис.1). Для проведения стресс – теста мы получаем выбранные значения из списков, помещая их в переменные selectedMaterial, selectedCrossSection, selectedTestResult (предварительно явно преобразовав их к соответствующему типу перечисления, т.к. списки возвращаю значение типа object).

Далее мы используем переменную типа StringBuilder, чтобы собрать результат стресс-теста. Значения переменных selectedMaterial, selectedCrossSection, selectedTestResult с помощью методв ToString() мы преобразуем в строки.

```
materialListBox.ItemsSource = Enum.GetValues(typeof(Material)).Cast<Material>();
crossSectionListBox.ItemsSource = Enum.GetValues(typeof(CrossSection)).Cast<CrossSection>();
testResultListBox.ItemsSource = Enum.GetValues(typeof(TestResult)).Cast<TestResult>();
```

Рисунок 1. Загрузка перечислений в ListBox`ы

```
Material selectedMaterial = (Material)materialListBox.SelectedItem;
CrossSection selectedCrossSection = (CrossSection)crossSectionListBox.SelectedItem;
TestResult selectedTestResult = (TestResult)testResultListBox.SelectedItem;

StringBuilder selectionStringBuilder = new StringBuilder();

selectionStringBuilder.Insert(selectionStringBuilder.Length, "Material: ");
selectionStringBuilder.Append(selectedMaterial.ToString());


selectionStringBuilder.Insert(selectionStringBuilder.Length, ", CrossSection: ");
selectionStringBuilder.Append(selectedCrossSection.ToString());

selectionStringBuilder.Insert(selectionStringBuilder.Length, ", CrossSection: ");
selectionStringBuilder.Append(selectedTestResult.ToString());


resultLabel.Content = selectionStringBuilder;
```

Рисунок 2. Код формирования результата стресс-теста

Задание 2: В этом задании мы используем структуру с двумя полями:

- Result: TestResult;
- ReasonForFailure: string;

которые содержат результат проведения стресс теста и причину неудачного результата (если поле Result имеет значение Pass, полю ReasonForFailure не будет присвоено значение, и в нём будет установлено значение по умолчанию конструктором – для строк – null, т.к. мы не определили свой конструктор – значения полей будем определять после инициализации структуры (конструктор по умолчанию для структуры определить нельзя)).

Для проведения стресс-тестов мы создали класс TestManager и определили такое его действие (метод), как GenerateResult

```
ссылка: 1
public TestCaseResult[] GenerateResult(TestCaseResult[] TestCaseResults_arr)
{

    Random rand = new Random();
    passCount = 0;
    failCount = 0;

    for (int i = 0; i < TestCaseResults_arr.Length; i++)
    {


        //Присваиваем значение полю Result и полю ReasonForFailure_arr

        int a = rand.Next(0, 21);

        arr[i] = a;

        if (a > 10)
        {
            TestCaseResults_arr[i].Result = TestResult.Pass;
            passCount = passCount + 1;
        }
        else if (a < 10)
        {
            TestCaseResults_arr[i].Result = TestResult.Fail;
            failCount = failCount + 1;
            TestCaseResults_arr[i].ReasonForFailure = reasonForFailure_arr[rand.Next(0, 3)];
        }
    }
```

Рисунок 3. Код метода GenerateResult

```
Ссылок: 5
public enum TestResult { Pass, Fail }
// Structures Exercise 2
/// <summary>
/// Structure containing test results
/// </summary>
Ссылок: 4
public struct TestCaseResult
{
    /// <summary>
    /// Test result (enumeration type)
    /// </summary>
    public TestResult Result;
    /// <summary>
    /// Description of reason for failure
    /// </summary>
    public string ReasonForFailure;
}
```

Рисунок 4. Определение структры TestCaseResult

Представлены 2 проекта, реализованных в Visual Studio Common Eddition 2019. Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.