Міністерство освіти, науки, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»

Об’єктно-орієнтоване програмування.

Частина 1.

Основи об’єктно-орієнтовного програмування.

М Е Т О Д И Ч Н І В К А З І В К И

до виконання комп’ютерного практикуму

Київ – 2014

Міністерство освіти, науки, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»

Об’єктно-орієнтоване програмування.

Частина 1.

Основи об’єктно-орієнтовного програмування.

М Е Т О Д И Ч Н І В К А З І В К И

до виконання комп’ютерного практикуму

для студентів напряму підготовки
6.050701 "Електротехніка та електротехнології"

Рекомендовано Вченою радою
факультету електроенерготехніки та автоматики

Київ
НТУУ "КПІ"
2014

Об’єктно-орієнтоване програмування. Частина 1. Основи об’єктно-орієнтовного програмування. Методичні вказівки до виконання комп’ютерного практикуму для студентів напряму підготовки бакалаврів 6.050701 „Електротехніка та електротехнології” // Уклад.: Д.В. Настенко, В.В. Заколодяжний – К.: НТУУ “КПІ”, 2014.– 73 с.

Гриф надано Вченою радою ФЕЕА НТУУ „КПІ”

(Протокол № __ від __/__/2014 р.)

Навчальне видання

Об’єктно-орієнтоване програмування.

Частина 1.

Основи об’єктно-орієнтовного програмування

Методичні вказівки

до виконання комп’ютерного практикуму

для студентів напряму підготовки

6.050701 „Електротехніка та електротехнології”

Укладачі: ***НАСТЕНКО*** *Дмитро Васильович*, ст. викл.

ЗАКОЛОДЯЖНИЙ *Володимир Васильович*, асистент

Відповідальний редактор *О.С. Яндульський*, професор, д.т.н.

Рецензент *Т.Л. Кацадзе*, канд. техн. наук

Зміст

Вступ

Дисципліна "Обчислювальна техніка та алгоритмічні мови 3" є складовою частиною в фаховій підготовці бакалавра з напряму підготовки 6.090615 "Електротехніка та електротехнології".

Дисципліна знайомить студентів із сучасним методами та принципами розробки програмного забезпечення ПК: операційною системою Windows, інтегрованим середовищем розробки Visual Studio .NET, об'єктно-орієнтованим програмуванням та мовою C#. Вивчення курсу супроводжується значною кількістю занять комп'ютерного практикуму, що дозволяє студентам отримати стійкі практичні навички по застосуванню програмних засобів, що вивчаються. Дисципліна "Обчислювальна техніка та алгоритмічні мови" є необхідною для успішного вивчення подальших курсів, пов'язаних з управлінням сучасною енергетикою.

Дисципліна відноситься до циклу дисциплін природничо-наукової підготовки. Дисципліни, які забезпечують "Обчислювальну техніку та мови програмування", наступні: "Вища математика", "Фізика", "Теоретичні основи електротехніки" та "Англійська мова".

Загальний навчальний час (включаючи СРС), необхідний для вивчення дисципліни диференціюється відповідно до планів спеціальностей, вказаних на титульному листі.

Дисципліна викладається студентам спеціальностей "Системи управління виробництвом та розподілом електроенергії" у п'ятому, шостому та сьомому семестрах.

Метою вивчення дисципліни є отримання знань з основ та принципів об'єктно-орієнтованого програмування та методів взаємодії прикладних програм з різноманітними пристроями релейного захисту.

Вивчення матеріалу дисципліни дозволяє отримати знання:

- основ об'єктно-орієнтованого програмування;
- взаємодії програм з пристроями релейного захисту;
- методів вимірювання, обробки і аналізу результатів дослідження;
- основ теорії натурного експерименту;
- тенденцій розвитку науки і техніки.

Уміння:

- проектувати та розроблювати об'єктно-орієнтовані програми;
- користуватися сучасними мовами програмування та середовищами розробки;
- аналізувати і обробляти результати експерименту;
- самостійно орієнтуватися в літературі по програмуванню.

Даний методичний посібник розроблено згідно програми п'ятого семестру.

Угоди по оформленню коду

Найменування ідентифікаторів. Загальні правила

Це необов'язкові правила, але їх використання рекомендоване, бо спрощує розуміння вашого коду, чи розуміння вами чужого коду.

1. Пам'ятайте! Код частіше читається, ніж пишеться, тому не економте на зрозумілості та чистоті коду заради швидкості набору.
2. Не використовуйте малозрозумілі префікси чи суфікси (наприклад, венгерську нотацію), сучасні мови та засоби розробки дозволяють контролювати типи даних на етапі розробки та збирання.
3. Не використовуйте підкреслювання для розділення слів усередині ідентифікаторів, це подовжує ідентифікатори та утруднює читання. Замість цього використовуйте стиль найменування Кемел чи Паскаль.
4. Намагайтеся не використовувати скорочення лишній раз, пам'ятайте про тих, хто читає код.
5. Намагайтеся робити імена ідентифікаторів якомога коротше (але не на шкоду читабельності). Пам'ятайте, що сучасні мови дозволяють формувати ім'я з просторів імен і типів. Головне, щоб зміст ідентифікатора був зрозумілий у використовуваному контексті. Наприклад, кількість елементів колекції краще назвати Count, а не CountOfElementsInMyCollection.
6. Коли придумуєте назву для нового, загальнодоступного (public) класу, простору імен або інтерфейсу, намагайтеся не використовувати імена, що потенційно або явно конфліктують зі стандартними ідентифікаторами.
7. Переважно використовуйте імена, які ясно і чітко описують призначення або сенс сутності.
8. Намагайтеся не використовувати для різних сутностей імена, які відрізняються тільки регістром літер. Компоненти, що розробляються, можуть бути використані з мов, що не розрізняють регістр, і деякі методи (або навіть весь компонент) виявляться недоступними.
9. Намагайтеся використовувати імена з простим написанням. Їх легше читати і набирати. Уникайте (в розумних межах) використання слів з подвійними літерами, складним чергуванням приголосних. Перш, ніж зупинитися у виборі імені, переконайтеся, що воно легко пишеться і однозначно сприймається на слух. Якщо воно важко читається, і ви помиляєтеся при його наборі, можливо, варто вибрати інше.

Стилі використання регістру букв

- Паскаль – це значить, що перша буква велика, та всі наступні перші літери слів теж великі. Наприклад, **B**ack**C**olor, **L**ast**M**odified, **D**ate**T**ime.
- Кемел – це значить, що перша буква рядкова, а інші перші літери слів великі. Например, **b**order**C**olor, **a**ccess**T**ime, **t**emplate**N**ame.

Скорочення

1. Не використовуйте абревіатури або неповні слова в ідентифікаторах, якщо тільки вони не є загальноприйнятими. Наприклад, пишіть GetWindow, а не GetWin.

2. Не використовуйте акроніми, якщо вони не загальноприйняті в області інформаційних технологій.

3. Широко поширені акроніми використовуйте для заміни довгих фраз. Наприклад, *UI* замість User Interface, чи *Olap* замість On-line Analytical Processing.

4. Якщо є ідентифікатор довжиною менше трьох літер, що є скороченням, то його записують великими літерами, например System.**IO,** System.Web.**UI.** Імена довше двох букв записуйте в стилі Паскаль чи Кемел, наприклад, Guid, Xml, xmlDocument.

Приклад:

```
using System.IO;
using System.Web.UI;

public class Math
{
   public const PI = ...;
   public const E  = ...;
}
```

Не використовуйте імена, що співпадають з глобальними просторами імен, такими, як *System* та *Microsoft.* Намагайтеся, по можливості, уникати співпадінь з назвами інших сутностей .NET Framework.

1. Заняття №1.

Знайомство з Visual Studio .NET. Типи даних. Ввід та вивід

Мета роботи – знайомство з середовищем розробки Visual Studio на базі платформи .Net, базовими типами даних мови C#, методами вводу та виводу даних.

1.1. Теоретичні відомості

.Net Framework – це інтегрована в Windows платформа, для створення програмних додатків різних типів, серед яких Windows-додатки, Web-додатки, додатки для мобільних телефонів і т.д. При розробці .Net Framework переслідувалися наступні цілі:

- створення узгодженого середовища виконання програм в незалежності від того де вони розміщені на локальному комп'ютері чи в Інтернеті;
- мінімізація конфліктів між різними версіями програмного забезпечення;
- гарантування безпеки виконання стороннього коду;
- забезпечення єдиних принципів розробки програм для різних типів додатків;
- створення середовища розробки для незалежних від платформи гетерогенних додатків.

Два основних компоненти .Net Framework, це незалежне від мов програмування середовище виконання (Common Language Runtime, CLR) та базова бібліотека класів .NET.

Visual Studio .NET – інструмент для швидкої розробки різних типів програм на платформі .NET.

Мова C# (читається як *Сі шарп*) – одна з мов для написання додатків для архітектури .NET. Ця мова об'єктно-орієнтована, тому код програми в ній складається не з процедур та функцій, а з класів.

Найпростіша програма на мові C# виглядатиме так:

```
class SimpleProgram
{
   static void Main()
   {
   }
}
```

де `Program` – ім'я класу, `Main()` – метод, точка входу в програму, з якої починається виконання коду додатку. Метод з ім'ям Main в програмі може бути лише один. Фігурні дужки `{}` в мові C# виділяють набір операторів, або *блок*.

В середині класу або методу можна оголошувати змінні:

Тип_змінної Ім'я_змінної [=початкове_значення];

Наприклад:

```
int i;          //ціла змінна
double d = 1.5; //дійсна змінна з початковим значенням 1,5
```

За допомогою подвійних косих (//) задаються коментарі до кінця рядка. А за допомогою пар символів /* та */ – багаторядкові коментарі.

Область видимості змінної в C# – блок коду (обмежений фігурними дужками {}). Змінна створюється на вході до області видимості і не доступна при виході з неї. По виходу з області видимості змінна потрапляє в чергу на видалення до "збирача мусору" (Garbage collector, GC).

Необхідно зазначити, що усі типи даних в C# діляться на два основні види – типи-значення (Value-type) та типи-посилання (Reference-type). Також існує тип-вказівник (Pointer-type) але дані типи можна використовувати лише в "небезпечному" режимі, і в даному курсі вони не розглядаються.

От список вбудованих типів-значень C#:

Тип C#	**Тип .Net**	**Кількість байт**	**Діапазон значень**
`bool`	Boolean	1	true або false
`byte`	Byte	1	від 0 до 255
`sbyte`	Sbyte	1	від -128 до 127
`short`	Int16	2	від -32 768 до 32 767
`ushort`	UInt16	2	від 0 до 65 535
`int`	Int32	4	від -2 147 483 648 до 2 147 483 647
`uint`	UInt32	4	від 0 до 4 294 967 295
`long`	Int64	8	від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
`ulong`	UInt64	8	від 0 до 18 446 744 073 709 551 615
`float`	Single	4	наближено від $\pm 1.5\times10^{-45}$ до $\pm 3.4\times10^{38}$ з сімома значущими цифрами
`double`	Double	8	наближено від $\pm 5\times10^{-324}$ до $\pm 1.7\times10^{308}$ з 15 або 16 значущими цифрами
`decimal`	Decimal	12	наближено від $\pm 1.0\times10^{-28}$ до $\pm 7.9\times10^{28}$ з 28 значущими цифрами
`char`	Char	2	16-бітний символ Unicode

Зауваження: При створенні константи типу float після числа треба дописати «f» або «F» (1.2F), decimal - «m» або «M». Шістнадцяткові константи починаються з префікса «0x» (0xA – шістнадцяткове A, десяткове 10).

Як видно з назви, типи-посилання містять не самі дані, а посилання на фактичні дані. Таких вбудованих типів-посилань у мові C# усього два: `object` та `string`. Усі змінні типів-посилань можна називати об’єктами.

Оператори C#:

Оператор	**Опис**
=	присвоєння
+	додавання
-	віднімання
*	множення
/	ділення (для цілих операндів – цілочисельне ділення)
%	залишок від ділення (для цілих)
==	дорівнює
!=	не дорівнює
>	більше
<	менше
>=	більше або дорівнює
<=	менше або дорівнює
&&	логічне І (AND)
||	логічне Або (OR)
!	логічне Ні (Not)
&	побітове І
|	побітове Або
^	побітове виключне Або (XOR)
~	побітове Ні
<<	побітовий зсув вліво (SHL)
>>	побітовий зсув вправо (SHR)
+=	додати до змінної (i+=10 збільшує змінну і на 10)
-=	відняти від змінної
*=	помножити змінну
/=	поділити змінну
++x	префіксний інкремент (збільшити змінну у виразі на 1)
--x	префіксний декремент
x--	постфіксний інкремент (збільшити змінну після обчислення виразу на 1)
x++	постфіксний декремент
&=	побітове І
|=	побітове Або
^=	побітове виключне Або (XOR)
~=	побітове Ні
?:	тернарний оператор (схожий на if then else, A?B:C – означає обчислити значення виразу A, якщо воно істинне, то повернути значення B, інакше повернути значення C)

Приклади:

```
int i = 1;
int j;
j = i++;  //після обчислень i рівне 2, а j=1

int i = 1;
int j;
j = ++i;  //після обчислень i рівне 2 і j=2
```

Для виконання більш складних математичних обчислень використовуються поля та методи класу **Math** в просторі імен **System.**

Простором імен називають логічну структуру в додатках .NET, яка дозволяє групувати споріднені класи та розв'язувати можливі конфлікти імен класів.

Приклад:
Скласти програму для обчислення змінної за заданою формулою:

$$\Delta = \frac{tga - \lg^2 b}{g}$$

Ознайомившись з класом **Math** за допомогою помічника, знайдемо методи для знаходження тангенсу та натурального логарифму – `Math.Tan` та `Math.Log`.

Програма для обчислення цього виразу (без передачі результату) виглядатиме так:

```
class MathProgram
{
    static void Main(string[] args)
    {
        double alpha = 4.5;
        double beta = 3.2;
        double gamma = 2.3;

        double log = System.Math.Log(beta);
        double delta =(System.Math.Tan(alpha)-log * log)/gamma;
    }
}
```

Оператор **using** дозволяє вказати простір імен з яким ми працюємо, та напряму звертатися до класів цього простору. Оператори **using** завжди розміщуються на початку файлу програми, перед усіма іншими операторами. Враховуючи це, спростимо наш приклад.

```
using System;
class MathProgram
{
    static void Main(string[] args)
    {
        double alpha = 4.5;
        double beta = 3.2;
        double gamma = 2.3;

        double log = Math.Log(beta);
        double delta = (Math.Tan(alpha) - log * log) / gamma;
    }
}
```

Тепер розглянемо, як організувати ввід та вивід. В консольному додатку найпростіше скористатися методами класу **Console** простору імен **System.**

- **Console.Read()** – зчитати окремий символ з клавіатури (консолі). Повертає число int, т.т. його результат потрібно привести до типу char, щоб отримати символ:

```
char ch = (char)Console.Read();
```

- **Console.ReadLine()** – читання рядку символів:

```
string st = Console.ReadLine();
```

Для того, щоб перетворити отриманий рядок у число, можна скористатися об'єктом **Convert** простору імен **System**:

```
double d = Convert.ToDouble(Console.ReadLine());
```

- **Console.Write()**та **Console.WriteLine()**– використовуються для організації форматованого виводу на екран. Причому **Write** – виводить рядок символів на екран, а виводить **WriteLine** рядок символів з переводом виводу на новий рядок:

```
Console.WriteLine("Hello World!");
```

Щоб вивести на екран значення якоїсь змінної, можна скористатися форматованим виводом. Наприклад:

```
int i = 1234;
Console.WriteLine("i = {0,5}", i);
```

Де `{0,5}` форматує значення виразу або змінної, що виводиться. Перше число у фігурних дужках відповідає за порядковий номер змінної у списку змінних перерахованих через кому (***нумерація починається з 0***). Наступне число – відповідає за кількість позицій на екрані виділених для відображення значення змінної (в наведеному прикладі – 5).

Для виводу чисел з плаваючою крапкою (комою), можна вказати і кількість цифр після крапки. Наприклад:

```
double d = 1234.12345;
Console.WriteLine("d = {0,7:f3}.",d);
```

Виведе на екран:

```
d = 1234,123.
```

Тут `f` – символ форматування, вказує що число треба виводити, як дійсне з плаваючою крапкою, а число 3 – що дробова частина числа повинна містити три цифри.

Всього символів форматування вісім:

Символи форматування	**Опис**
`F або f`	Для виводу чисел з фіксованою точністю
`E або e`	Для виводу чисел в експоненційному форматі
`P або p`	Для виводу процентів
`N або n`	Для виводу чисел з розділеними розрядами
`C або c`	Для виводу значень чисел в національних валютах
`D або d`	Для виводу чисел в десятковому форматі
`G або g`	Звичайний спосіб - вивід чисел з фіксованою точністю або в експоненційному форматі, вибирається коротший варіант
`X або x`	Для виводу цілих шістнадцяткових чисел

1.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом (література, помічник F1 та бібліотека MSDN).
2. Запустити Visual Studio. Пуск/Програми/Microsoft Visual Studio/ Microsoft Visual Studio, або за допомогою ярлику на робочому столі.
3. Створити в Visual Studio новий проект File/New/Project…/ Visual C#/ ConsoleApplication та задати йому нове ім’я.
4. Ознайомитися з поняттями "Project" ("проект") та "Solution" ("рішення") у Visual Studio.
5. За допомогою помічника (Help – клавіша F1) ознайомитися з полями та методами класу **`Math.`**
6. Вибрати індивідуальне завдання. Номер варіанту відповідає номеру студента у списку групи.
7. Скласти і налагодити програму на мові C# , яка реалізує введення вихідних даних, обчислення значення змінної за заданою формулою, виведення результатів у зручній формі на екран.
8. При виводі на екран обчислених значень, дослідити як працюють всі різноманітні формати.
9. Підготувати звіт по роботі. Вимоги викладені у розділі "Зміст звіту з лабораторної роботи" у кінці цих методичних вказівок (стор. 71).

1.3. Індивідуальні завдання

Скласти алгоритм і програму для обчислення значення змінної за заданою формулою**:**

1.	$Y = \sqrt{d + rc\cos}(wt + d)$	16.	$x=U^2+\lg t \cos(t^2-\cos rt)$
2.	x=1+ arctg (t-cos t)	17.	$x=U^2+arctg(t^4-\cos st)$
3.	$z = e^{bt-5}w\frac{r}{h}$	18.	$I = \frac{1}{R}(1-e^{\frac{R}{L}t})$
4.	$Y = \sqrt{f\cos}(wt+1)$	19.	$Y = \sqrt{gt\cos}(wt + j)$
5.	$b = \frac{1-tg(x)}{l}$	20.	$z = e^{bt-u}\sin(\frac{u}{h})$
6.	$w=U^2\sin x +\cos(t^4-\cos st)$	21.	$x=U^2+\cos(t^2-\cos rt)$
7.	$x=1+ arctg (t^2-\cos kt)$	22.	$A = \sqrt{1-k^2\sin^2 j}$
8.	$Y = \frac{k}{\ln(x+d)^4}$	23.	$A = \frac{\sin kt}{(x+rc)^4}$
9.	$G = \sqrt{\cos t - \sin^2 tj}$	24.	$Y = \sqrt{gt^2 + s\cos}(wt + j)$
10.	$A = \sqrt{1-k^2\sin^2 j}$	25.	$R = e^{\sqrt{1+\sin y}}$
11.	$Y = \frac{k}{\ln(x+d)^4}$	26.	$A = \sqrt{1-k^2\sin^2 j}$
12.	$L = \sqrt{gt^2 + S^{\cos(wt)}}$	27.	$S = \sqrt{P\cos j + jQ\sin j}$
13.	$Q = \sqrt{\cos t - h^2\sin^2 j}$	28.	$A = \sqrt{\cos t - k^2\sin^2 j}$
14.	$b = \frac{1-ctg^2(x)}{l}$	29.	$\Delta = \frac{1-\ln^2(x)}{q}$
15.	$u = \frac{1}{c}(1-e^{\frac{r}{c}t})$	30.	$I = \frac{1}{r}(1-e^{\frac{r}{L}t})$

1.4. Контрольні запитання.

1. Яке призначення .Net Framework?
2. Що таке Visual Studio .NET?
3. Що таке "solution" ("рішення") та "project" ("проект") у Visual Studio?
4. Які вбудовані типи даних існують в C#?
5. Які основні поля та методи класу **`Math`**?
6. За допомогою яких методів організується ввід та вивід?
7. Що таке форматований вивід?
8. Які оператори C# ви знаєте?

2. Заняття №2.

Робота з перерахуваннями. Умовні оператори

Мета роботи – знайомство з поняттям перерахувань та умовними операторами мови C#.

2.1. Теоретичні відомості

Перерахування – дозволяють створити в програмі набір констант. Вони являють собою типи зі значеннями. Спрощений синтаксис перерахувань наступний:

```
enum <Ім_я_перерахування> {список_констант}
```

де `список_констант` – перераховані через кому константи, що входять до перерахування.

Наприклад:

```
enum Days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

є перерахуванням днів тижня, де понеділку `Mon` ставиться у відповідність число 0, `Tue` – 1 (вівторок), і т. д. Якщо ж ми хочемо понеділку поставити у відповідність – 1, вівторку – 2 і т.д., то перерахування матиме вигляд:

```
enum Days {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
```

де значення першого елементу встановлено в 1, а решта елементів приймають значення на 1 більше ніж попередній елемент.

Аналогічно можна задавати значення для кожного елементу перерахування. При цьому, якщо два елементи перерахування матимуть однакові числові значення, то вони вважатимуться рівними.

Для доступу до елементів перерахування використовується *оператор крапка*. Наприклад для звертання до вівторка: **`Days.Tue`**. Якщо ж нам потрібно за елементом перерахування отримати його числове значення, то можна скористатися оператором явного перетворення типів.

```
Console.WriteLine((int)Days.Wed);
```

Тут оператор **`(int)`** – вказу на те, що значення виразу треба привести до цілочисельного типу **`int`**. В результаті, на екран виведеться число 3.

Всі перерахування в C# зв'язані з базовим класом `Enum`, для якого передбачено ряд корисних методів. Наприклад, якщо нам потрібно дізнатися ім'я елементу перерахування за його значенням, можна скористатися методом `GetName`:

```
using System;
class ProgramWithEnum
{
    enum Days { Mon=1, Tue, Wed, Thu, Fri, Sat, Sun };
    static void Main(string[] args)
    {
        Console.WriteLine(Enum.GetName(typeof(Days),3));
    }
}
```

Ця програма виведе на екран назву третього дня тижня – середи: `Wed`.

Умовний оператор дозволяє програмі перевірити логічну умову, і в залежності від результату перевірки, виконати той чи інший фрагмент коду. Синтаксис умовного оператору наступний:

```
if (<Умова>)
    оператор1;
[else
    оператор2;]
```

Якщо `Умова` прийме значення `true`, то виконується перший оператор або набір операторів, інакше другий оператор або набір операторів. Причому друга частина умовного оператору **`else`** … – не є обов'язковою.

При використані набору операторів вони об'єднуються в блок, та обмежуються фігурними дужками, тоді синтаксис умовного оператору набуває наступного вигляду:

```
if (<Умова>){
    оператори1
}
[else{
    Оператори2
}]
```

Інший оператор для керування логікою програми в C# – це оператор **`switch`**. Він дозволяє керувати ходом програми при скінченому наборі варіантів вибору, що зарані відомі. Синтаксис цього оператору наступний:

```
switch (<вираз>){
       case <значення1>:
           оператори1
           break;
       [[case <значення2>:
           оператори2
           break;]
         . . .
       default:
           оператори_за_замовчуванням
           break;]
}
```

Тут тип виразу може бути одним з типів, що можна перерахувати (байт, символ, ціле і т.д.) або рядком. Якщо значення виразу співпаде зі значенням значення1, то виконається набір операторів оператори1, зі значенням значення2, то виконається набір операторів оператори2. Якщо ж не підійде жоден з перераховних варіантів, то виконається набір операторів оператори_за_замовчуванням після ключового слова **default**.

Наприклад:

```
int i = 1;
switch (i){
    case 1:
        Console.WriteLine("Один");
        break;
    case 2:
        Console.WriteLine("Два");
        break;
    default:
        Console.WriteLine("Інше число");
        break;
}
```

Наступний приклад показує як для різних значень виконати одні й ті самі дії:

```
int i = 2;
switch(n){
    case 1:
    case 2:
    case 3:
        Console.WriteLine("Це 1, 2 або 3");
        break;
default:
    Console.WriteLine("Інше число");
    break;
}
```

Щоб моделювати більш складні дії в операторі **switch**, можна замінити оператор **break** на **goto**, наприклад:

```
int d = 1, k=1;
switch (d){
    case 1:
        k++; break;
    case 2:
        k += 3;
        goto case 1;
}
```

Ключове слово **`default`** – необов'язкове в операторі **`switch`**.

2.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Вибрати індивідуальне завдання. Номер варіанту відповідає номеру студента у списку групи.
3. За допомогою помічника (Help – клавіша F1) та Інтернету ознайомитися з властивостями та методами класу **`Enum.`**
4. Виконати завдання використовуючи можливості перерахувань, а також за допомогою оператора **`switch`**.
5. Налагодити програму на комп'ютері.
6. Підготувати звіт по роботі.

2.3. Індивідуальні завдання

Скласти програму на мові C#:

1. Написати програму, що дозволяє одержати словесне найменування традиційних оцінок (2 – незадовільно, 3 – задовільно, 4 – добре, 5 – відмінно).
2. Написати програму, яка за номером місяця виводить на екран назву пори року.
3. Написати алгоритм, що за номером дня тижня – цілому числу від 1 до 7 повинен видавати як результат кількість пар у групі у відповідний день.
4. Для кожної цифри вивести її назву (0 – нуль, 1 – один, 2 – два і т.д.).
5. В залежності від того чи введена відкрита дужка або закрита, надрукувати "відкрита кругла дужка" або "закрита фігурна дужка". (Ураховувати круглі, квадратні, фігурні дужки).
6. Залежно від уведеного символу L, S, V програма повинна обчислювати довжину кола; площу кола; об'єм шару.
7. Скласти програму, що залежно від порядкового номера дня тижня (1, 2, ..., 7) виводить на екран його назву (понеділок, вівторок, ..., неділя).
8. Скласти програму, що залежно від порядкового номера місяця (1, 2, ..., 12) виводить на екран кількість днів у цьому місяці. (Для лютого вивести рядок '28 або 29').
9. Скласти програму, що залежно від введеного номіналу видавала б назву банкноти української валюти (1 – одна гривня, 2 – дві гривні, 5 – п`ять гривень і т.д.).
10. Мастям гральних карт умовно привласнені наступні порядкові номери: «піки» – 1, «трефи» – 2, «бубни» – 3, «чирви» – 4. По заданому номеру масті m ($1 < m < 4$) визначити назву відповідної масті.
11. Скласти програму, що залежно від введеного номіналу видавала б назву монети української валюти (1 – одна копійка, 2 – дві копійки, 5 – п`ять копійок і т.д.).

12. Гральним картам умовно привласнені наступні порядкові номери залежно від їхнього достоїнства: «валетові» – 11, «дамі» – 12, «королеві» – 13, «тузу» – 14. Порядкові номери інших карт відповідають їхнім назвам («шістка», «дев'ятка» і т.п.). За заданим номером карти k (6 < k < 14) визначити достоїнство відповідної карти.
13. Скласти програму, що залежно від порядкового номера місяця (1, 2, ..., 12) виводить на екран його назву (січень, лютий, ..., грудень).
14. Написати програму, яка б за введеним номером одиниці виміру (1 – кілограм, 2 – міліграм, 3 – грам, 4 – тонна, 5 – центнер) і масі M видавала б відповідне значення маси в кілограмах.
15. Написати програму, що дозволяє по останній цифрі числа визначити останню цифру його квадрату.
16. Для кожної введеної цифри (0-9) вивести відповідне їй назву англійською мовою (0 – zero, 1 – one, 2 – two, ...).
17. Залежно від уведеного символу L, S, V програма повинна обчислювати периметр квадрату; площу квадрату; об’єм куба.
18. Скласти програму, що за введеною літерою A, B, C, D, E, F виведе назву традиційної оцінки (A – відмінно, B і C – добре, D і E – задовільно, F – незадовільно).
19. Написати програму, що за номером місяця видає назву наступного за ним місяця (при m = 1 отримуємо лютий, при m = 4 – травень).
20. Написати програму, яка б за введеним номером пори року (1 – зима, 2 – весна, 3 – літо, 4 – осінь) видавала відповідній цій порі року назву місяців, кількість днів у кожному з місяців.
21. Для цілого числа від 1 до 99 надрукувати фразу «Мені k років», за умови, що при деяких значеннях k слово «років» треба замінити на слово «рік» або «роки». Наприклад, 11 років, 51 рік, 22 роки.
22. Написати програму, яка б за введеним номером одиниці виміру (1 – дециметр, 2 – кілометр, 3 – метр, 4 – міліметр, 5 – сантиметр) і довжині відрізка L видавала б відповідне значення довжини відрізка в метрах.
23. Написати програму, що за введеним числом від 1 до 5 (номеру курсу) видає відповідне повідомлення «Привіт, k-курсник». Наприклад, якщо k=1, «Привіт, першокурсник»; при k=4: «Привіт, четвертокурсник».
24. Є пронумерований список деталей: 1) шуруп, 2) гайка, 3) гвинт, 4)цвях, 5) болт. Скласти програму, що за номером деталі виводить на екран її назву.
25. Написати програму, що за номером місяця видасть кількість святкових і вихідних днів у цьому році.
26. Скласти програму, що дозволяє по останній цифрі даного числа визначити останню цифру куба цього числа.
27. Визначити, чи є введена буква українського алфавіту голосною.
28. Визначити, чи є введена буква українського алфавіту приголосною.

2.4. Контрольні запитання

1. Що таке перерахування?
2. Який клас .Net Framework пов’язаний з перерахуваннями?
3. Для чого призначений умовний оператор if?
4. Наведіть приклади використання оператора switch?

3. Заняття №3.

Робота з циклами

Мета роботи – ознайомитись з різними типами циклів в C#. Навчитися застосовувати цикли в програмних додатках.

3.1. Теоретичні відомості

Цикли – програмні конструкції, що дозволяють виконувати один той самий блок коду до тих пір, поки не виконається якась умова. В мові C# передбачено чотири види циклів:

- `for`
- `while`
- `do ... while`
- `foreach ... in`

Перші три перейшли в мову C# з класичного C, а **`foreach`** – Visual Basic.

Цикл **`for`** – найбільш універсальний з циклів у мовах C, C++ та Java, з його допомогою можна змоделювати роботу всіх інших. Синтаксис цього циклу:

```
for([ініціалізатор];[умова];[ітератор])
      оператор;
```

де `Ініціалізатор` – вираз або набір виразів, перерахованих через кому, що обчислюється перед початком циклу. Зазвичай використовується для задання початкових значень змінних, що використовуються в циклі.

`Умова` – логічний вираз, значення якого перевіряється перед кожною операцією цикла, якщо значення `true`, то виконується тіло циклу, якщо ж `false` – цикл завершується.

`Ітератор` - вираз або набір виразів, перерахованих через кому, що обчислюються в кінці кожної ітерації циклу.

Слід зазначити, що всі три віщенаведені елементи циклу – необовязкові. І найпростіший цикл, має вигляд:

```
for (;;){
          Оператор //... набір операторів
}
```

де `Оператор` – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклади:

```
for (int i = 1; i < 10; i++){
   Console.WriteLine(i);
}
```

цикл, що виводить на екран значення від 1 до 9. Зауважимо, що змінна **i** оголошена в середині циклу, тому її область видимості обмежена тілом циклу. Необхідно слідкувати, щоб поза межами циклу не було змінних з таким самим ім'ям, інакше може виникнути непорозуміння.

```
int i, j;
for (i = 1, j = 2; i < 10 && j < 24; i++, j+=2){
    Console.WriteLine(i*j);
}
```

цикл, де одночасно використовуються два лічильники.

Цикл **while** – класичний цикл з передумовою, має наступний синтаксис:

```
while (<умова>)
    оператор;
```

де Умова – повертає значення true або false. Це значення обчислюється та перевіряється перед кожною ітерацією, якщо значення true – то виконується тіло циклу, якщо false – цикл завершується.

Оператор – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int n = 1;
while (n < 5){
    Console.WriteLine("n = {0}", n);
    n++;
}
```

Виводить на екран рядки:

```
n = 1
n = 2
n = 3
n = 4
```

Цикл **do ... while** – цикл з післяумовою. Аналогічний до **while** за виключенням того, що умова перевіряється в кінці кожної ітерації, а не перед початком. Має наступний синтаксис:

```
do
   оператор;
while (<умова>);
```

де Умова – повертає значення true або false. Це значення обчислюється та перевіряється наприкінці кожної ітерації, якщо значення true – то ще раз виконується тіло циклу, якщо false – цикл завершується.

Оператор – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int n = 1;
do {
    Console.WriteLine("n = {0}", n);
    n++;
}
while (n < 0);
```

Виводить на екран рядок:

```
n = 1
```

Звідки стає помітним, що, так як умова перевіряється у кінці, то тіло циклу обов'язково виконається хоча б один раз.

Цикл **foreach...in** – спеціальний цикл для роботи з набором елементів – колекцією (прикладом одного з типів колекцій є масив, про масиви і колекції детальніше на наступних заняттях). Має такий синтаксис:

```
foreach(<тип ім'я_зміної> in <вираз_колекція>)
     оператор;
```

де ім'я_зміної – ім'я змінної, яка використовується для доступу перебором до всіх елементів, заданих за допомогою виразу_колекції.

тип – тип даних змінної.

Оператор – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int[] myArray = { 1, 3, 5, 8, 2 };
foreach (int i in myArray) {
     Console.Write("i = " + i + ", ");
}
```

виведе на екран:

```
i = 1, i = 3, i = 5, i = 8, i = 2
```

Іноді виникає необхідність припинити виконання циклу за якоюсь умовою в тілі циклу, для цього в мові C# передбачені оператори **break** та **continue.**

`break` – перериває виконання циклу і передає управління до наступного за циклом оператору.

`continue` – перериває поточну ітерацію циклу і переходить одразу до наступної.

3.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Вибрати індивідуальне завдання. Номер варіанту відповідає номеру студента у списку групи.
3. Виконати завдання використовуючи цикли.
4. Перевірити роботу операторів `break` та `continue`. Навести свої приклади використання цих операторів.
5. Налагодити програму на комп'ютері.
6. Підготувати звіт по роботі.

3.3. Індивідуальні завдання

1. Обчислити суму членів ряду:

$$Z = \frac{x}{(m+2)!} + \frac{x^2}{(m+3)!} + \frac{x^3}{(m+4)!} + \ldots$$

з точністю до члена ряду, що менше *E=0.0001*.

2. Обчислити суму членів ряду:

$$z = \cos x + \frac{\cos 2x}{4} + \frac{\cos 3x}{9} + \ldots + \frac{\cos nx}{n^2} + \ldots$$

з точністю до *0,0001*.

3. Обчислити суму членів ряду:

$$z = 1 + x^2 + \frac{x^4}{2!} + \ldots + \frac{x^{2n}}{n!} + \ldots$$

з точністю до члена ряду, що менше *E=0.00001*.

4. Обчислити суму членів ряду:

$$y = \frac{1}{2 \cdot 3} + \frac{2}{3 \cdot 4} + \ldots + \frac{n}{(n+1)(n+2)} + \ldots$$

з точністю до члена ряду, що менше 10^{-4}.

5. Обчислити суму членів ряду:

$$z = \frac{\sin x}{4} + \frac{\sin 2x}{9} + \frac{\sin 3x}{16} + \ldots + \frac{\sin nx}{(n+1)^2} + \ldots$$

з точністю до члена ряду, що менше *E*.

6. Обчислити суму членів ряду:

$$z = 2\left[\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \ldots + \frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}} + \ldots\right]$$

з точністю до члена ряду, що менше 10^{-6}.

7. Обчислити суму членів ряду:

$$y = x^n + \frac{1}{2}x^{n-1} + \ldots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \ldots$$

з точністю до члена ряду, що менше 10^{-6}.

8. Обчислити значення функції exp з точністю до п'ятого знака, використовуючи розкладення в ряд Тейлора:

$$e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 \ldots$$

9. Обчислити значення функції cos з точністю до п'ятого знака, використовуючи розкладення в ряд Тейлора:

$$\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 \ldots$$

10. Обчислити суму членів ряду:

$$z = x^2 - \frac{x^4}{3!} + \frac{x^6}{5!} + \frac{x^8}{7!} + \ldots$$

з точністю до члена ряду, що менше E.

11. Обчислити з точністю до п'ятого знака:

$$\log_e x = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \ldots$$

з точністю до члена ряду, що менше E.

12. Обчислити суму членів ряду:

$$y = \frac{2}{1\cdot 4} + \frac{3}{2\cdot 5} + \ldots + \frac{n+1}{n(n+3)} + \ldots$$

з точністю до члена ряду, що менше 10^{-4}.

13. Обчислити *arctg* використовуючи наступне розкладання в ряд:

$$arctg\frac{1}{x} = \frac{1}{x} - \frac{1}{3x^3} + \frac{1}{5x^5} - \mathbf{L}\,.$$ для $X>1$

Обчислення провести з точністю до шостого знака.

14. Обчислити значення функції sin з точністю до п'ятого знака, використовуючи розкладення в ряд Тейлора:

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 \ldots$$

15. Обчислити суму членів ряду:

$$y = \frac{1}{2}x + \frac{2}{3}x^2 + \ldots + \frac{n}{(n+1)}x^n + \ldots,$$

з точністю до члена ряду, що менше 10^{-4}.

16. Обчислити суму членів ряду:

$$z = \frac{1}{m!} + \frac{mx}{(m+1)!} + \frac{m(m-1)}{(m+2)!}x^2 + \frac{m(m-1)(m-2)}{(m+3)!}x^3 + \ldots$$

з точністю до члена ряду, що менше 10^{-4}. Для визначеного поточного члена ряду використовувати рекурентну формулу: $y_n = y_{n-1}\frac{x(m-n+1)}{m+n}$, де n - номер члена ряду. Початкове значення y взяти рівним $\frac{1}{m!}$.

17. Обчислити суму членів ряду:

$$z = 1 - \frac{x^2}{2} + \frac{x^4}{4} - \frac{x^6}{6} + \ldots$$

з точністю до члена ряду, що менше E.

18. Скласти програму обчислення відрізка ряду: $S = \sum_{i=0}^{n}(-1)^i \frac{x^{2i+1}}{2^i + 1}$;

x = 0,56. Розрахунки припинити, якщо $x^{2i+1} \Big/ (2^i + 1) \le 10^{-4}$.

19. Скласти програму обчислення функції $W = \sqrt[3]{Z}$, використовуючи ітераційну формулу: $W_{n+1} = W_n + \frac{1}{3}(\frac{Z}{W_n^2} - W_n)$.

Розрахунки припинити, якщо $|W_{n+1} - W_n| < 10^{-5}$; $W_0 = 6$; $Z = 220$.

20. Обчислити $y = \sqrt{x}$ з точністю E, використовуючи співвідношення:

$$y_{i+1} = \frac{1}{2}(y_i + \frac{x}{y_i}); \quad E = 10^{-3}; \quad x = 8{,}6; \quad y_0 = 2{,}8 \ .$$

21. Знайти значення змінної $y = \frac{\sum_{k=1}^{\infty}\frac{1}{k^2}}{arctg(x^2 - 1)}$ для кожного із значень x, розташованих на інтервалі $(-2; 2)$ з кроком $0{,}5$, з точністю до $0{,}0001$. Роздрукувати таблицю значень y і x.

22. Знайти значення змінної $y = \frac{1}{\sum_{i=1}^{\infty} \frac{1}{i^2} \sin x}$ з точністю до *0,00001* для значень *x*, розташованих на інтервалі *(0; 1)* з кроком *0,25*. Роздрукувати таблиці значень *y* і *x*.

23. Обчислити значення змінної $p = \frac{1}{x^2+1} \prod_{k=1}^{\infty} \frac{1}{(k+x)}$ з точністю *E* для значень *x*, розташованих на інтервалі *(-1; 2)* з кроком *0,25*. Роздрукувати таблицю значень *p* і *x*.

24. Знайти значення змінної $y = \frac{\sum_{k=1}^{\infty} \frac{1}{k^2}}{\cos^2(x^2 - 0.5)}$ з точністю до *0,00001* для значень *x*, розташованих на інтервалі *(0; 1)* з кроком *0,25*; на інтервалі *(0; 3)* з кроком *0,5*. Роздрукувати таблицю значень *y* і *x*.

25. Знайти методом Ньютона корінь рівняння $f(x) = e^x$-cos(2x) з точністю до 10^{-4}, взявши як перше приблизне значення кореня *x=0*. Наступні наближення знаходяться за формулою: $x_{s+1} = x_s - \frac{f(x)}{f'(x)}$, де *f `(x)* - похідна від функції *f(x)*.

26. Обчислити $y = \sqrt{x}$ з точністю до четвертого знака, використовуючи співвідношення: $y_{i+1} = \frac{1}{2}(y_i + \frac{x}{y_i})$; $x = 9,5$; $y_0 = 1,3$.

3.4. Контрольні запитання

1. Які цикли використовуються в мові C#?
2. В чому відмінність між циклами **`while`** та **`do ... while`**?
3. Опишіть роботу циклу **`for`**?
4. Які оператори використовуються для керування роботою циклів?

4. Заняття №4.

Робота з масивами. Сортування та пошук в масивах

Мета роботи – знайомство з поняттям масивів. Навчитися працювати з одновимірними та багатовимірними масивами C#.

4.1. Теоретичні відомості

Масив – набір елементів однакового типу, звертатися до яких можна за одним спільним ім'ям. При оголошенні масиву використовують наступний синтаксис:

```
<тип>[] <ім'я_масиву>;
```

де `тип` – тип даних, `ім'я_масиву` – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Після оголошення масиву необхідно за допомогою оператору **new** задати його розмірність.

Наприклад:

```
int[] myArray;
myArray = new int[10]; //масив цілих з десяти елементів

double[] NumArr = new double[5];//масив з 5-ти дійсних чисел
```

Можна також одразу ініціювати масив початковими значеннями:

```
int[] myArray = new int[5] { 1, 3, 5, 8, 2 };
```

або ще простіше:

```
int[] myArray = { 1, 3, 5, 8, 2 };
```

без ключового слова **new** розмір масиву визначиться автоматично.

Звертатися до елементів масиву можна виказавши його ім'я та індекс елемента у квадратних дужках. Проте слід зазначити, що у мові C# ***нумерація елементів починається з нуля***, і перший елемент масиву матиме індекс 0.

Всі масиви C# є нащадками класу **System.Array** у якого є ряд корисних властивостей та методів. Наприклад властивість:

Length – повертає кількість елементів масиву;

Rank – повертає розмірність масиву (1 для одновимірного і т.д.).

Тепер розглянемо приклад програми для виводу на єкран масиву за допомогою циклу **for**:

```
using System;
class ProgramWithArray {
    static void Main() {
        int[] arr = { 1, 3, 5, 8, 2 };
        for (int i = 0; i < arr.Length; i++) {
            Console.WriteLine("arr[{0}]={1}", i, arr[i]);
        }
        Console.ReadKey();
    }
}
```

Ця програма виведе на екран наступне:

```
arr[0]=1
arr[1]=3
arr[2]=5
arr[3]=8
arr[4]=2
```

В цій програмі можна замінити цикл **for** на **foreach**, тоді цикл виглядатиме так:

```
foreach (int i in arr) {
    Console.WriteLine(i);
}
```

але слід враховувати, що у випадку циклу **foreach** ітераційну змінну можна використовувати тільки для читання.

Серед методів **System.Array** слід виділити наступні:

Clear() – очищує мсив, заповнивши всі елементи 0 або пустими значеннями **null**;

Copy() – копіює заданий діапазон значень з одного масиву в інший;

Clone() – створює повну копію масиву разом зі значеннями;

IndexOf(), LastIndexOf() – відповідно шукають перше та останнє входження елементу в масив;

Sort() – сортує одновимірний масив за зростанням;

BinarySearch() – швидкий пошук в сортованому масиві;

Reverse() – змінює порядок слідування елементів масиву на протилежний.

Розглянемо приклад:

```
using System;
class Program {
    static void Main() {
        int[] arr = { 1, 8, 5, 3, 2 };       //створимо масив
        int[] arrNew = (int[])arr.Clone();   //створимо копію
        Array.Sort(arrNew);                  //відсортуємо масив
        for (int i = 0; i < arrNew.Length; i++) {
            //виведемо відсортовані значення
            Console.WriteLine("arrNew[{0}]={1}", i, arrNew[i]);
        }
        Console.ReadKey();
    }
}
```

результатом роботи програми буде:

```
arrNew[0]=1
arrNew[1]=2
arrNew[2]=3
arrNew[3]=5
arrNew[4]=8
```

Багатовимірний масив – це масив який визначається двома або більше вимірами, а до його елементів треба звертатися за домомогою двох або більше індексів.

Найпростіші багатовимірні масиви – двовимірні прямокутні масиви (або матриці). Для їх оголошення використовують наступний синтаксис:

```
<тип>[,] <ім'я_масиву>;
```

де `тип` – тип даних, `ім'я_масиву` – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Приклад створення матриці 5 на 7:

```
int[,] matrix = new int[5, 7];
```

до її елементів можна звертатися наступним чином:

```
matrix[0,0] = 7;  //лівий верхній елемент
matrix[3,5] = 10; //десь всередині
matrix[4,6] = 0;  //правий нижній елемент
```

Тепер розглянемо як заповнити та вивести на екран прямокутну матрицю за допомогою циклів:

```
using System;
class Program {
    static void Main() {
        int[,] matrix = new int[5, 7];  //створюємо матрицю
        //заповнюємо елементи матриці сумою індексів
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 7; j++) {
                matrix[i, j] = i + j;
            }
        }
        //виведемо отримані значення на екран
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 7; j++) {
                Console.Write("{0,3}",matrix[i, j]);
            }
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

На екрані отримаємо:

```
  0  1  2  3  4  5  6
  1  2  3  4  5  6  7
  2  3  4  5  6  7  8
  3  4  5  6  7  8  9
  4  5  6  7  8  9 10
```

Аналогічно до двовимірних можна створювати тривимірні, чотиривимірні і т.д. масиви:

```
int[, ,]matrix3 = new int[5, 7, 5];
int[, , ,]matrix4 = new int[4, 4, 3, 3];
```

Зубчасті масиви **(**jagged array**)** – це масиви рядки яких можуть мати різну довжину. Причому кожен рядок зубчастого масиву є одновимірним масивом. Ці масиви оголошуються на ступним чином:

```
тип[][] ім'я_масиву;
```

Щоб створити двовимірний зубчастий масив, спочатку треба задати кількість рядків, наприклад:

```
int[][] arrJ = new int[3][];
```

потім задати розмірність кожного з рядків:

```
arrJ[0] = new int[3];
arrJ[1] = new int[4];
arrJ[2] = new int[2];
```

звертатися до елементів масиву потрібно так:

```
arrJ[0][0] = 2;
arrJ[0][1] = 4;
//...
arrJ[2][1] = 17;
```

Часто зубчасті масиви доцільніше використовувати замість прямокутних. Наприклад, якщо в матриці необхідно переставляти рядки, то при використанні зубчастого масиву відпадає необхідність поелементного копіювання даних.

Приклад:

```
using System;
class ProgramWithJaggedArray {
    static void Main() {
        //створюємо зубчасту матрицю 3x4
        float[][] arrJ = new float[3][];
        for (int i = 0; i < arrJ.Length; i++) {
            arrJ[i] = new float[4];
        }
        //заповнюємо елементи рядка його номером
        for (int i = 0; i < arrJ.Length; i++) {
            for (int j = 0; j < arrJ[i].Length; j++) {
                arrJ[i][j] = i;
            }
        }
        //міняємо місцями перший рядок з останнім
        float[] x = arrJ[0];
        arrJ[0] = arrJ[2];
        arrJ[2] = x;
    }
}
```

4.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. За допомогою помічника (Help – клавіша F1) та Інтернету ознайомитися з властивостями та методами класу **`Array.`**
3. Вибрати індивідуальне завдання. Номер варіанту відповідає номеру студента у списку групи.
4. Налагодити програму на комп’ютері.
5. Забезпечити вивід результатів роботи програми на екран комп’ютера.
6. Підготувати звіт по роботі.

4.3. Індивідуальні завдання

Скласти алгоритм і програму розв’язання задач

1. Нормувати квадратну матрицю, поділивши усі елементи кожного рядка на найбільший за модулем елемент цього рядка.
2. Знайти найбільший елемент матриці A (20x30) і номер рядка та стовпця, у яких він знаходиться.
3. Задано одновимірний масив A. Знайти мінімальний за абсолютним значенням і відмінний від нуля елемент; утворити новий одновимірний масив B , щоб його елементи визначалися як ціле від ділення відповідного елемента масиву A на знайдений елемент.
4. У прямокутній матриці переставити стовпці так, щоб сума елементів стовпця зростала від першого стовпця до останнього.
5. У квадратній матриці переставити рядки так, щоб кількість ненульових елементів у рядках зростала від першого рядка до останнього.
6. Є дві квадратні матриці. Поміняти місцями їх головні діагоналі, спочатку упорядкувавши елементи діагоналей за зростанням.
7. Обчислити суму елементів матриці A (20x20), розміщених над головною діагоналлю.
8. Знайти найбільші елементи кожного рядка матриці X(20x20) і записати їх у масив Y.
9. Визначити середнє арифметичне додатних елементів кожного стовпця матриці X(25x25) за умови, що у кожному стовпці є хоча б один додатний елемент.
10. Обчислити суми елементів кожного рядка матриці X (20x20), визначити найменше значення цих сум і номер відповідного рядка.
11. Визначити кількість додатних і від'ємних елементів матриці A (10x15).
12. Визначити кількість додатних елементів кожного стовпця матриці A (10x20) і запам’ятати їх у масиві M.
13. Знайти найбільший елемент матриці X (15x20) і записати нулі в той рядок та стовпець, де він знаходиться.
14. Написати програму множення матриці на вектор стовпчик.
15. Переписати перші елементи кожного рядка матриці A (15x25), які більші ніж C, у масив B. Якщо в рядку немає елемента, який більше за C, записати нуль в масив B.
16. Визначити мінімальні елементи кожного рядка матриці X(20x20) і помістити їх на головну діагональ, а діагональні елементи записати на місце мінімальних.
17. Обчислити суму елементів матриці C(25x25), які розміщені під головною діагоналлю.
18. Обчислити суму всіх непарних елементів матриці D(30x25), які розміщені в парних рядках.
19. Обчислити суму всіх парних елементів матриці R(25x20), які розміщені в непарних рядках.

20. Визначити середнє арифметичне всіх непарних елементів кожного парного стовпця матриці X(25x30).
21. Є дві матриці. Підрахувати кількість елементів, які більше за абсолютним значенням ніж число S (задається довільно).
22. Записати (+1) замість максимального елемента масиву ($x_1,x_2,...,x_{50}$), а (-1) - замість мінімального.
23. Знайти суму додатних елементів матриці над допоміжною діагоналлю.
24. Для масиву ($a_1,a_2,...,a_{80}$) обчислити найбільше і найменше значення модуля різниці між сусідніми елементами.
25. Знайти суму від'ємних елементів матриці під допоміжною діагоналлю.
26. Обчислити суми елементів парних рядків матриці X (20x20), визначити найменше значення цих сум і номер відповідного рядка.
27. Дано два одновимірних масиви довільної розмірності. Знайти суму максимальних елементів кожного масиву.
28. Дано два одновимірних масиви довільної розмірності. Знайти суму мінімальних елементів кожного масиву.

4.4. Контрольні запитання

1. Що так масив?
2. Які типи масивів Ви знаєте?
3. Назвіть властивості та методи класу **`System.Array`**?
4. Як задаються зубчасті масиви?
5. В чому переваги різних типів масивів?

5. Заняття №5.

Класи, поля та методи. Створення найпростіших класів

Мета роботи – знайомство з поняттями клас та інкапсуляція. Створення найпростіших класів з полями та методами.

5.1. Теоретичні відомості

Клас – це шаблон який визначає форму, зміст та поведінку об'єкту. Об'єкт – це екземпляри класу. За допомогою класу реалізується перший з основних принципів об'єктно-орієнтованого програмування – ***інкапсуляція***. Інкапсуляція – це об'єднання в одному цілому даних та алгоритмів обробки цих даних. Цей термін також включає в себе приховування даних, тобто приховування від зовнішнього користувача деталей реалізації об'єкта. Дані зберігаються у вигляді ***полів***, а алгоритми обробки у вигляді ***методів***. Поля, ще іноді називають змінними класу. Оголошення класу в мові C# наступне:

```
[модифікатор_доступу] class <ім'я_класу> {
   // оголошення полів
   [мод._доступу] <тип1> <ім'я_поля1>
        [=початкове_значення1];
   [мод._доступу] <тип2> <ім'я_поля2>;
        [=початкове_значення2]
   // ...
   // оголошення методів
   [мод._доступу] <тип_значення1> <ім'я_методу1>
          ([параметри1]){
       //тіло методу
   }
   [мод._доступу] <тип_значення2> <ім'я_методу2>
          ([параметри2]){
       //тіло методу
   }
   //...
}
```

де `[модифікатор_доступу]` – дозволяє вказувати рівень доступу до класу або елементів класу, приймає значення з наступної таблиці:

Модифікатор	Опис
`private`	компонент доступний тільки в середині класу, приймається за замовчуванням
`protected`	компонент доступний тільки в середині класу або в класі нащадку
`internal`	компонент доступний тільки в середині класу або в цій же програмі (або зборці)
`protected internal`	компонент доступний тільки в середині класу, в класі нащадку або в цій же програмі (або зборці)
`public`	компонент доступний без обмежень

Оголошення змінних класів або полів ми вже розглядали на попередніх заняттях, зазначимо тільки, що при створенні об’єкту, якщо не задані початкові значення, поля отримають наступні початкові значення:

Тип поля	Значення за замовчуванням
Всі числові типи	0
bool	false
char	‘\0’
Об’єктні типи	null – що означає пусте посилання

Приклад класу, який складається тільки з полів:

```
public class Transformer {          //клас трансформатор
    public int windingCount = 2;    //кількість обмоток
    public int temperature;          //поточна температура
    public string power;            //потужність
    public string model;            //марка
    public int yearBuilt;           //рік випуску
}
```

Більш детально зупинимося на описі методів:

тип_значення1, тип_значення2 - тип змінної, що повертається методом. Якщо метод ніого не повертає – цей тип задається ключовим словом **void**, інакше тіло циклу обов’язково повинно містити оператор **return**, за допомогою якого метод повертає значення заданого типу. Якщо тип об’єктний, то повертається посилання (вказівник) на цей об’єкт.

параметри1, параметри2 – необов’язковий список параметрів вигляду:

```
[ref|out|params] <тип_параметру1> <ім’я_параметру1> [, <тип_параметру2> <ім’я_параметру2>[, ...]]
```

наприклад:

```
public class Power {        //клас електрична потужність
    //... поля якщо потрібно
    // метод обчислює повну потужність
    public float Full(float U, float I) {
        return U * I;
    }
    // метод для обчислення активної потужності
    public float Active(float U, float I, float cosFi) {
        return Full(U, I) * cosFi;
    }
}
```

Створення об'єкту. Для описаного класу `Transformer` об'єкт створюємо за допомогою ключового слова **`new`**.

```
Transformer myTrans1 = new Transformer();
```

Тепер ми можемо звертатися до полів та методів створеного об'єкту використовуючи оператор «крапка» (.) разом з посиланням на об'єкт:

```
myTrans1.windingCount = 2;
myTrans1.model = "АТДЦТН-400/330/110/35";
myTrans1.yearBuilt = 2003;
Console.WriteLine("Марка={0}", myTrans1.model);
```

В цьому прикладі `Transformer()` після ключового слова **`new`** – це ***конструктор класу*** (метод, що виконується при створенні об'єкту).

Типи передачі параметрів. Для простих типів передача параметрів може бути за значенням та за посиланням, для об'єктних тільки за посиланням. При ***передачі за значенням***, в методі створюється копія змінної-параметра в яку записується значення і при зміні значення в середині методу значення в базовій змінній не змінюється. Для того, щоб змінювати значення в середині методу використовується ***передача за посиланням***.

За замочуванням передача простих параметрів відбувається за значенням. Для передачі простих параметрів за посиланням використовується ключове слово **`ref`**. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class Test_ref {
    void Mov(int a, int b) {
        int x = a;
        a = b;
        b = x;
    }
    void Xchg(ref int a, ref int b) {
        int x = a;
        a = b;
        b = x;
    }
```

```
    static void Main(string[] args) {
        int a = 3, b = 4;
        Test_ref p = new Test_ref();
        p.Mov(a, b); //значення a, b не змінюються
        Console.WriteLine("a={0}, b={1}", a, b);
        p.Xchg(ref a, ref b); //a та b обмінялися значеннями
        Console.WriteLine("a={0}, b={1}", a, b);
        Console.ReadKey();
    }
}
```

Виведе на екран наступне:

```
a=3, b=4
a=4, b=3
```

Для передачі простих параметрів за значеням або за допомогою **ref** їм обовязково задавати початкові значення.

Для більшості випадків від методів достатньо, щоб вони повертали одне значення, щоб отримати з методу додаткові значення – використовуються ***вихідні параметри***. Вихідні парамери оголошуються за допомогою ключового слова **out**, і вимагають обов'язкової ініціалізації в середині методу. Приклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class TestOutParameter {
    void Power(int c, out int a, out int b) {
        a = c * c;
        b = c * c * c;
    }
    static void Main(string[] args) {
        int a, b, c = 5;
        TestOutParameter p = new TestOutParameter();
        p.Power(c, out a, out b);
        Console.WriteLine("a={0}, b={1}", a, b);
        Console.ReadKey();
    }
}
```

Виведе на екран значення квадрату та кубу числа 5:

```
a=25, b=125
```

Щоб мати можливість передати в метод, змінну кількість параметрів можна використати ключове слово **params.** Воно може використовуватись тільки один раз і для останнього параметру, вказуючи, що метод може мати будь-яку кількість параметрів цього типу.

```
using System;
using System.Collections.Generic;
using System.Text;

class Test_Params {
    int Sum(params int[] a) {
        int s = 0;
        foreach(int i in a) {
            s += i;
        }
        return s;
    }
    static void Main(string[] args) {
        Test_Params p = new Test_Params();
        Console.WriteLine("Sum={0}", p.Sum(1, 2, 3, 4));
        Console.WriteLine("Sum={0}", p.Sum(1, 2, 3, 4, 5));
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
Sum=10
Sum=15
```

Перевантаження методів – дозволяє використовувати одні й ті самі імена методів змінюючи набір аргументів. Це корисно коли схожі дії треба виконати для різних типів даних, або для різних наборів даних. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class OverloadTest {
        int Add(int a, int b) {         //для додавання цілих
        return a + b;
    }
    double Add(double a, double b) {    //для додавання дійсних
        return a + b;
    }
    int Add(int a) {                    //для збільшення на 1
        return ++a;
    }
    static void Main(string[] args) {
        OverloadTest t = new OverloadTest ();
        Console.WriteLine("Sum= {0}", t.Add(1, 2));
        Console.WriteLine("Sum= {0}", t.Add(3.4, 4.4));
        Console.WriteLine("Sum= {0}", t.Add(7));
        Console.ReadKey();
    }
}
```

Виведе:

```
Sum= 2
Sum= 7,8
Sum= 8
```

5.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Навести свій приклад класу пов’язаний з майбутньою спеціальністю.
3. У класу повинно бути декілька полів різних типів.
4. Також клас повинен містити декілька методів з різними типами аргументів.
5. В цьому ж класі навести приклади перевантаження методів.
6. Налагодити програму на комп’ютері.
7. Забезпечити вивід результатів роботи програми на екран комп’ютера.
8. Підготувати звіт по роботі.

5.3. Контрольні запитання

1. Що таке клас, інкапсуляція?
2. З чого складається клас?
3. Які існують модифікатори рівня доступу?
4. Навести приклади різних типів передачі параметрів?
5. Що таке перевантаження? До якого з основних принципів об’єктно-орієнтованого програмування перевантаження має безпосереднє відношення?

6. Заняття №6.

Використання властивостей та статичних методів

Мета роботи – подальше знайомство з класами. Навчитися використовувати конструктори та деструктори, властивості та статичні методи.

6.1. Теоретичні відомості

Конструктор – це спеціальний метод класу, що викликається при створенні об'єкту класу. Ім'я конструктора завжди повинне співпадати з іменем класу. Якщо розробником класу явно не визначено жодного конструктора, то для класу автоматично створюється конструктор без аргументів. Конструктори не повертають жодних значень, тому при їх оголошенні не потрібно вказувати тип, навіть **void**. Приклад:

```
public class Car {          //клас машина
public string color;        //колір
public string model;        //модель
public int yearBuilt;       //рік випуску

public Car() {              //конструктор без аргументів
    Console.WriteLine("Створюється машина");
    }
}
```

Для описаного класу `Car` об'єкт створюємо за допомогою ключового слова **new**.

```
Car myCar = new Car();
```

При цьому при створенні об'єкту `myCar` буде визвано, явно заданий конструктор `Car()`.

Як і для інших, методів для конструкторів можливе перевантаження, тобто можна створювати конструктори з різними наборами аргументів. Найчастіше конструктори використовують, що б задавати початкові значення полів. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;
public class Car {              //клас машина
    public string color;        //колір
    public string model;        //модель
    public int yearBuilt;       //рік випуску

    public Car() {             //конструктор без аргументів
        Console.WriteLine("Створюється машина");
    }
```

```
    public Car(string c, string m, int yb) {
        color = c;              //конструктор з аргументами
        model = m;
        yearBuilt = yb;
    }
}
class Car_Test {
   static void Main(string[] args) {
      Car c0 = new Car();
      Car c1 = new Car("Black", "BMW", 2000);
      Console.ReadKey();
    }
}
```

Ключове слово **this** – використовується для посилання на поточний об'єкт. Одне з застосувань, те що воно дозволяє оголошувати аргументи з тими ж назвами, що і поля. Так конструктор з попереднього прикладу можна переписати наступним чином:

```
    public Car(string color, string model, int yearBuilt) {
        this.color = color;
        this.model = model;
        this.yearBuilt = yearBuilt;
    }
```

тут, `this.color` - означатиме звертання до поля `color`, а просто `color` - до аргументу конструктора `color`. Також ключове слово **this** можна використовувати при посиланні на інший конструктор при перевантаженні, цей механізм дозволяє запобігати надмірному дублюванню коду програми.

```
    public Car() {
        Console.WriteLine("Створюється машина");
    }
    public Car(string color, string model, int yearBuilt)
        : this() {
        this.color = color;
        this.model = model;
        this.yearBuilt = yearBuilt;
    }
```

В цьому прикладі, для конструктору з аргументами спочатку здійсниться виклик конструктору без аргументів. Інший приклад:

```
    public Car(string color, string model, int yearBuilt) {
        this.color = color;
        this.model = model;
        this.yearBuilt = yearBuilt;
    }

    public Car(string model, int yearBuilt)
        : this("", model, yearBuilt) {
    }
```

– демонструє, як на базі існуючого створити конструктор з меншою кількістю аргументів.

Властивості – це один з механізмів інкапсуляції, вони одночасно дозволяють встановлювати та зчитувати значення полів за допомогою методів та приховувати поля від користувача. Властивості також зручно використовувати для перевірки введених даних на відповідність.

Для властивостей можна одночасно визначати два методи **get** та **set** (**get –** для отримання значення, **set –** для встановлення) або по одинці кожен з них. Приклад:

```
public class Car {
    private string model;

    public string Model {
        get { return model; } //зчитує значення поля model
        set { model = value; }//встановлює значення поля model
    }
}
class Car_PropertyTest {
    static void Main(string[] args) {
        Car c0 = new Car();
        c0.Model = "Audi";
        Console.WriteLine(c0.Model);
        Console.ReadKey();
    }
}
```

При компіляції, ці методи перетворяться на **get_Model** та **set_Model** відповідно, тому при написанні програм не рекомендується створення методів з подібними іменами.

Якщо визначено тільки метод **get** – то поле доступне тільки для читання, якщо **set** – то тільки для запису.

Статичні поля, методи та властивості – задаються за допомогою ключового слова **static,** та належать самому класу, а не конкретному об’єкту. Тому і звертання до них виконується **за ім’ям класу**, а не за ім’ям об’єкту. Прикладами статичних полів та методів можуть слугувати поля та методи вже знайомого нам класу **Math** (такі як `Math.PI`, `Math.Abs(x)`, `Math.Pow(x, y)` і т. д.).

Деструктор – це спеціальний метод класу, що викликається при знищенні об’єкту. Оголошується за ім’ям класу та за допомогою символу **~**, без модифікаторів. Може використовуватись, наприклад, для закриття пов’язаних з об’єктом файлів, або для підрахунку створених об’єктів класу.

```
public class Car {
    public static int numOfCars = 0; //кількість машин
    public Car(){                    //конструктор
        ++numOfCars;
    }
    ~Car(){                          //деструктор
        --numOfCars;
    }
}

public delegate void Calc(double x, double y);
class Car_ConstructorTest {
    static void Main(string[] args) {
        Car c0 = new Car();
        Car c1 = new Car();
        Console.WriteLine(Car.numOfCars);
        Console.ReadKey();
    }
}
```

Статичний конструктор – оголошується за допомогою ключового слова **static**, без модифікаторів та не має аргументів. Викликається один раз перед створенням першого екземпляру класу. Може використовуватись для ініціалізації статичних полів та різного типу налаштувань класу. Приклад:

```
    static Car() {
        Console.WriteLine("Static Car");
    }
```

6.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Доповнити приклад з попереднього заняття кількома конструкторами.
3. Для полів створити відповідні властивості.
4. Використати статичні поля, властивості та методів.
5. Налагодити програму на комп’ютері.
6. Забезпечити вивід результатів роботи програми на екран комп’ютера.
7. Підготувати звіт по роботі.

6.3. Контрольні запитання

1. Що таке конструктор?
2. Наведіть приклади перевантаження конструкторів?
3. Коли використовується ключове слово **this**?
4. Що таке властивості? Для чого вони використовуються?
5. Чим відрізняються статичні компоненти від динамічних?
6. Що таке деструктор?
7. Які особливості статичного конструктора?

7. Заняття №7.

Вивчення успадкування та поліморфізму

Мета роботи – знайомство з двома основами об'єктно-орієнтованого програмування – успадкуванням та поліморфізмом.

7.1. Теоретичні відомості

Успадкування – це властивість класу породжувати нащадків. Тобто: клас-нащадок (дочірній клас) успадковує від базового(батьківського) частину полів і методів, та може доповнюватись новими полями та методами.

Поліморфізм – це властивість споріднених об'єктів вирішувати подібні проблеми різними способами.

Щоб один клас успадковував інший використовують наступний синтаксис:

```
class <ім'я_класу_нащадку> : <ім'я_базового_класу>
```

Для прикладу розглянемо базовий клас двовимірних графічних об'єктів:

```
public class Object2D {
    public int x, y; //поля-координати
    //конструктор
    public Object2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //метод для "відображення"
    public void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}",
          x, y);
    }
    //метод для "переміщення"
    public void Move(int dx, int dy) {
        x += dx;
        y += dy;
        Draw();
    }
}
```

маючи цей клас, не важко створити нові – крапку та коло:

```
// клас крапка
public class Point : Object2D {
    public Point(int x, int y)// конструктор Point посилається
        : base(x, y) {        // на батьківський конструктор -
                              // Object2D
        //тут може бути додатковий код
    }
}
```

тут ключове слово **base** означає, що при створенні об'єкту крапка спочатку викликається базовий конструктор Object2D, після чого в середині конструктора Point можна виконати унікальні для цього класу дії. Що наочніше демонструється в класі – коло.

```
// клас коло
public class Circle : Object2D {
    public int r; //нове поле - радіус
    public Circle(int x, int y, int r)
        : base(x, y) {
        this.r = r;
    }
}
```

В цьому класі введено додаткове поле для радіусу – r, і значення поля радіус – задається у власному конструкторі Circle після виконання батьківського конструктора Object2D.

Обидва ці класи автоматично успадковують методи Draw() і Move(int dx, int dy) батьківського класу, і результат роботи цих методів ні чим не відрізнятиметься від визову аналогічних методів для об'єкту Object2D.

Проте більш цікавим є випадок коли методи відображення (Draw()) для кола та точки відрізнятимуться від базових. При цьому при оголошенні батьківського метода повинно використовуватись ключове слово virtual (можливий), а при оголошенні методу для дочірнього класу ключове слово – override (відхиляти). Наведемо повний приклад:

```
using System;
using System.Collections.Generic;
using System.Text;
public class Object2D {
    public int x, y; //поля-координати
    //конструктор
    public Object2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //метод для "відображення"
    public virtual void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}",
          x, y);
    }
    //метод для "переміщення"
    public void Move(int dx, int dy) {
        x += dx;
        y += dy;
        Draw();
    }
}
```

```
// клас крапка
public class Point : Object2D {
    public Point(int x, int y)//конструктор Point посилається на
        : base(x, y) {        // батьківський конструктор -
                              // Object2D
    }
    public override void Draw() {
        //повністю перекриває батьківський метод
        Console.WriteLine("Крапка з координатами {0},{1}",
          x, y);
    }
}
// клас коло
public class Circle : Object2D {
    public int r; //нове поле - радіус
    public Circle(int x, int y, int r)
        : base(x, y) {
        this.r = r;
    }
    public override void Draw() {
        //спочатку викликає батьківський метод
        base.Draw();
        //потім доповнює його
        Console.WriteLine("це коло радіусом {0}", r);
    }
}
class UsingOverloadedObjects {
    static void Main() {
        Point p0 = new Point(10,20);
        Point p1 = new Point(1, 40);
        Circle c0 = new Circle(15, 17, 10);
        p0.Draw();
        p1.Draw();
        c0.Draw();
        Console.ReadKey();
    }
}
```

Ця програма виведе на екран наступну інформацію:

```
Крапка з координатами 10,20
Крапка з координатами 1,40
Об'єкт з координатами 15,17
це коло радіусом 10
```

Крім того, з усіма об’єктами нащадками базового класу можна спільно працювати, як з об’єктами базового класу. Для прикладу залишимо класи `Object2D`, `Point` та `Circle` без змін перепишемо тільки клас `UsingOverloadedObjects`:

```
class UsingOverloadedObjects {
    static void Main() {
        Point p0 = new Point(10,20);
        Point p1 = new Point(1, 40);
        Circle c0 = new Circle(15, 17, 10);

        //створено масиву вказівників на об’єкти Object2D
        Object2D[] objects = new Object2D[3];
        //спочатку всі елементи цього масиву рівні null
        //заповнимо його посиланнями на створені об’єкти
        objects[0] = p0;
        objects[1] = p1;
        objects[2] = c0;
        //тепер над ними можна виконувати групові дії
        //відображення
        foreach(Object2D o in objects) {
            o.Draw();
        }
        Console.WriteLine("\n\rпісля зміщення\n\r");
        //зміщення
        foreach(Object2D o in objects) {
            o.Move(5, -4);
        }
        Console.ReadKey();
    }
}
```

Результатом роботи цієї програми буде:

```
Крапка з координатами 10,20
Крапка з координатами 1,40
Об'єкт з координатами 15,17
це коло радіусом 10

після зміщення

Крапка з координатами 15,16
Крапка з координатами 6,36
Об'єкт з координатами 20,13
це коло радіусом 10
```

Слід зазначити, що взагалі всі класи .Net є нащадками базового класу **System.Object.** Це наслідування відбувається неявно, без буд-яких вказівок. Від **System.Object** всі класи успадковують ряд корисних методів, серед яких:

- `public Type` **`GetType`**`()` – повертає об’єкт класу `Type`, що містить детальну інформацію поточний об’єкт (до якого класу відноситься, чи є об’єкт масивом, чи являється екземпляром класу або перерахуванням і т. д.
- `public virtual string` **`ToString`**`()` – повертає представлення

об’єкту у вигляді рядка (для більності об’єктів – це ім’я класу, а для чисел значення у вигляді рядку), може перевантажуватись.

- `public virtual bool` **`Equals`**`(Object obj)` для порівняння поточного об’єкту з іншими), може перевантажуватись, повертає `true`, якщо об’єкти рівні.
- `public static bool` **`Equals`**`(Object objA, Object objB)` – для порівняння двох різних об’єктів.

7.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Вибрати варіант завдання згідно номеру в групі.
3. Побудувати ієрархію класів згідно варіанту.
4. Навести приклади перевантаження батьківських методів в класах-нащадках.
5. Налагодити програму на комп’ютері.
6. Забезпечити вивід результатів роботи програми на екран комп’ютера.
7. Підготувати звіт по роботі.

7.3. Індивідуальні завдання

1. Трансформатор, трансформатор струму, силовий трансформатор, автотрансформатор.
2. Студент, викладач, людина, доцент, декан.
3. Пошкодження, трифазне кз, двохфазне кз, однофазне кз на землю, двофазне кз на землю.
4. Держава, республіка, монархія, королівство, демократія.
5. Службовець, персона, робітник, інженер, головний інженер.
6. Деталь, механізм, виріб, вузол, гвинт.
7. Організація, компанія, банк, страхова компанія, енергетична компанія.
8. Газета, журнал, книга, видання, підручник.
9. Тест, іспит, випробування, залік, диференційний залік.
10. Їжа, продукт, товар, хліб, послуга.
11. Квитанція, накладна, документ, рахунок, чек.
12. Автомобіль, поїзд, транспорт, експрес, автобус.
13. Електричний двигун, двигун, двигун внутрішнього згоряння, дизель, реактивний двигун.
14. Тварина, риба, ссавець, птах, примат.
15. Корабель, пароплав, вітрильник, фрегат, атомохід.

7.4. Контрольні запитання.

1. Дайте означення понять успадкування та поліморфізм.
2. Який синтаксис успадкування у мові C# ?

3. Коли використовується ключове слово **`base`**?
4. За допомогою яких ключових слів реалізується механізм поліморфізму?
5. Яким чином можна з дочірнього класу можна звертатись до батьківських методів?
6. Який об’єкт є базовим для всіх класів в середовищі .Net?
7. Які методи є спільними для всіх класів?

8. Заняття №8.

Перевантаження операторів

Мета роботи – знайомство з механізмом перевантаження операторів.

8.1. Теоретичні відомості

Перевантаження операторів дозволяє дати визначення операторам для класів створених користувачем. Головна перевага механізму перевантаження операторів в тому, що він дозволяє використовувати природній синтаксис при написанні програм. Що приводить до зменшення витрат на написання програм.

Синтаксис перевантаження:

```
public static <тип_значення> operator <унарний_оператор>
(<тип_параметру> <назва_параметру>)

public static <тип_значення> operator <бінарний_оператор> (
      <тип_параметру1> <назва_параметру1>,
      <тип_параметру2> <назва_параметру2>
)
```

тут

`тип_значення` – тип результату дії оператора;

`унарний_оператор` – один з операторів **+, -, !, ~, ++, --, true, false**;

`бінарний_оператор` - один з операторів **+, -, *, /, %, &, |, ^, <<, >>, ==, !=, >, <, >=, <=**;

`тип_параметру` – тип параметру в операторі;

`назва_параметру` – тип параметру в операторі.

Приклад перевантаження:

```
using System;
public class Point{
    double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    // при додаванні крапок утворюється нова,
    // координати якої рівні сумі відповідних координат
    public static Point operator +(Point a, Point b){
        return new Point(a.x+b.x,a.y+b.y);
    }
    // при множенні крапки на число утворюється нова,
    // координати якої рівні добутку відповідних
```

```
    // координат на число
    public static Point operator *(Point a, double d) {
        return new Point(a.x * d, a.y * d);
    }
    // переозначимо метод ToString()
    public override string ToString() {
        return String.Format("Крапка({0},{1})", this.x, this.y);
    }
}
class UsingOperatorOverload {
    static void Main(string[] args) {
        Point p1 = new Point(5, 8);
        Point p2 = new Point(7, 12);
        Console.WriteLine(p1 + "+" + p2 + "=" + (p1 + p2));
        Console.WriteLine(p1 + "*" + 5 + "=" + (p1 * 5));
        Console.ReadKey();
    }
}
```

Ця програма виведе на екрані:

```
Крапка(5,8)+Крапка(7,12)=Крапка(12,20)
Крапка(5,8)*5=Крапка(25,40)
```

Те що в цій програмі перевантажено метод `ToString()`, дозволяє спростити вивід за допомогою методу `Console.WriteLine`, так як метод `ToString()` використовується за замовчуванням при перетворенні об'єкту в рядок.

Круглі дужки використоуються у виразах `(p1 + p2)` та `(p1 * 5)` для того, щоб спочатку виконались дії над об'єктами крапка, а вже потім перетворення в рядок за допомогою методу `ToString()`.

Слід зазначити, що оператори **`==`** та **`!=`**, **`>`** та **`<`**, **`>=`** та **`<=`** потрібно перевантажувати тільки парами.

Для прикладу розглянемо порівняння крапок:

```
    public static bool operator ==(Point a, Point b) {
        if(a.x == b.x && a.y == b.y) {
            return true;
        } else {
            return false;
        }
    }
    public static bool operator !=(Point a, Point b) {
        return !(a == b);
    }
```

Як бачимо з цього прикладу, при перевантаженні оператора "недорівнює", можна скористатися тим, що він обернений до оператора "дорівнює". Також можна спростити код оператора "дорівнює", скориставшись тим що вираз

`(a.x == b.x && a.y == b.y)` – повертає значення булівського типу:

```
    public static bool operator ==(Point a, Point b) {
        return a.x == b.x && a.y == b.y;
    }
```

Текст програми повністю:

```
using System;
public class Point{
    double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public override string ToString() {
        return String.Format("Крапка({0},{1})", this.x, this.y);
    }

    public static bool operator ==(Point a, Point b) {
        return a.x == b.x && a.y == b.y;
    }
    public static bool operator !=(Point a, Point b) {
        return !(a == b);
    }
}
class OperatorOverload {
    static void Main(string[] args) {
        Point p1 = new Point(5, 8);
        Point p2 = new Point(7, 12);
        Console.WriteLine(p1 + " та " + p2 +
            (p1 == p2 ? " - рівні " : " - не рівні"));
        Console.ReadKey();
    }
}
```

Результат роботи програми:

```
Крапка(5,8) та Крапка(7,12)- не рівні
```

Крім вищезгаданих операторів можна перевантажити оператор [], але для цього використовують спеціальну властивість – ***індексатор***. Індексатор можна використовувати для доступу до різних полів об'єкта або полів-масивів, оголошуються вони за допомогою ключового слова `this` та квадратних дужок `[]`.

Приклад використання індексатору для різних полів:

```
using System;
public class Car {              //клас машина
    public string color;        //колір
    public string model;        //модель
    public int yearBuilt;       //рік випуску
```

```
    // індексатор
    public string this[int i] {
        // властивість на читання
        get {
            switch(i) {
                case 0:
                    return model;
                case 1:
                    return color;
                case 2:
                    return yearBuilt.ToString();
                default:
                    return null;
            }
        }
        // властивість для запису
        set {
            switch(i) {
                case 0:
                    model=value;
                    break;
                case 1:
                    color=value;
                    break;
                case 2:
                    yearBuilt = Convert.ToInt32(value);
                    break;
                default:
                    break;
            }
        }
    }
}
class IndexatorOverload {
    static void Main(string[] args) {
        Car c = new Car();
        c[0] = "Audi";
        c[1] = "Green";
        c[2] = "2000";
        for(int i = 0; i < 3; i++) {
            Console.WriteLine(c[i]);
        }
        Console.ReadKey();
    }
}
```

На екрані буде наступне:

```
Audi
Green
2000
```

В цій програмі використання індексатора дозволяє поставити у відповідність кожному полю класа індекс і звертатися до полів по імені об’єкта за допомогою цього індекса.

Приклад використання індексатору для масиву:

```
using System;
public class Vector {
    int[] v;
    // конструктор
    public Vector(int size) {
        v = new int[size];
    }
    // метод
    public int Length() {
        if(v == null) {
            return 0;
        } else {
            return v.Length;
        }
    }
    // індексатор
    public int this[int i] {
        get { return v[i]; }
        set { v[i] = value; }
    }
}
class Program {
    static void Main(string[] args) {
        Vector vect = new Vector(5);
        for(int i = 0; i < vect.Length(); i++) {
            vect[i] = i * 2;
            Console.WriteLine("vect[{0}] = {1}", i, vect[i]);
        }
        Console.ReadKey();
    }
}
```

Програма виведе на екран:

```
vect[0] = 0
vect[1] = 2
vect[2] = 4
vect[3] = 6
vect[4] = 8
```

Як бачимо з цього прикладу, використання індексатору дозволяє звертатися до елементів поля масиву `v`, як до елеметів об’єкту `vect`.

8.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Вибрати варіант завдання згідно номеру в групі.
3. Написати програму на мові C#, яка б реалізувала вказані дії за допомогою перевантаження операторів.
4. Доповнити приклад перевантаженням методу `ToString()`.
5. Навести приклад використання індексатора.
6. Налагодити програму на комп'ютері.
7. Забезпечити вивід результатів роботи програми на екран комп'ютера.
8. Підготувати звіт по роботі.

8.3. Індивідуальні завдання

1. Додавання та порівняння комплексних чисел.
2. Додавання (поелементне) матриць.
3. Скалярний добуток векторів.
4. Множення та порівняння комплексних чисел.
5. Додавання векторів.
6. Ділення та порівняння векторів.
7. Множення матриць.
8. Порівняння векторів.
9. Віднімання (поелементне) матриць.
10. Множення вектора на число.
11. Віднімання та порівняння комплексних чисел.
12. Порівняння матриць.
13. Віднімання векторів.
14. Множення (поелементне) матриці на число.

8.4. Контрольні запитання.

1. Для чого потрібно перевантаження операторів?
2. Які оператори можна перевантажувати?
3. Яка особливість перевантаження операторів порівняння?
4. Що таке індексатор?
5. Які області застосування індексаторів?

9. Заняття №9.

Використання інтерфейсів

Мета роботи – знайомство з поняттям інтерфейсу. Навчитися створювати та використовувати інтерфейси.

9.1. Теоретичні відомості

Інтерфейси – це спеціальні класи, що містять тільки назви методів (без реалізації методів та без полів). Реалізацію ж, цих методів покладено на класи-нащадки інтерфейсів. Оголошують інтерфейси за допомогою ключового слова **interface.**

```
[модифікатор_доступу] interface <ім'я_інтерфейсу> {
        // оголошення методів
        [мод._доступу] <тип_значення1> <ім'я_методу1>
               ([параметри1]);
        [мод._доступу] <тип_значення2> <ім'я_методу2>
               ([параметри2]);
        }
        //...
    }
```

ім'я_інтерфейсу – зазвичай починається з великої літери **I** (використання великої літери **I** напочатку назви інтерфейсу не є обов'язковим, проте – бажане, для полегшення розуміння коду).

Основне застосування інтерфейсів – полегшить синхронізацію програмного коду написаного різними розробниками. Знову розглянемо приклад з графічними об'єктами. Нехай нам потрібно, щоб всі графічні об'єкти мали дві координати, відображалися та рухалися. Можемо створити інтерфейс:

```
using System;
public interface IGraphObject {
    // властивості для координат
    int X {
        get;
        set;
    }
    int Y {
        get;
        set;
    }

    void Draw();                    //метод для відображення
    void Move(int dx, int dy);      //метод для переміщення
}
```

Тепер успадкуємо від цього інтерфейсу клас Object2D:

```
public class Object2D : IGraphObject {
    protected int x, y; //поля-координати

    //конструктор
    public Object2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //реалізація успадкованих властивостей
    public int X {
        get { return x; }
        set { x = value; }
    }
    public int Y {
        get { return y; }
        set { y = value; }
    }
    //реалізація успадкованих методів
    //метод для відображення
    public virtual void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}",
            x, y);
    }
    //метод для переміщення
    public void Move(int dx, int dy) {
        x += dx;
        y += dy;
        Draw();
    }

}
```

Перевага інтерфейсі над класами, це можливість успадковувати один клас від кількох інтерфейсів. Наприклад опишемо інтерфейс для обертання IRotated:

```
public interface IRotated {
    //метод для обертання
    void Rotate(int x0, int y0, double angle);
}
```

Додамо його в список предків для Object2D і нам доведеться додатково описати метод Rotate.

```
public class Object2D : IGraphObject, IRotated {
    protected int x, y; //поля-координати
    //конструктор
    public Object2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

```
    //реалізація успадкованих властивостей
    public int X {
        get { return x; }
        set { x = value; }
    }
    public int Y {
        get { return y; }
        set { y = value; }
    }
    //реалізація успадкованих методів
    //метод для "відображення"
    public virtual void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}",
            x, y);
    }
    //метод для "переміщення"
    public void Move(int dx, int dy) {
        x += dx;
        y += dy;
        Draw();
    }
    //
    public void Rotate(int x0, int y0, double angle) {
        // метод що описує обертання
        // навколо заданого центру
        // ...
    }
}
```

Слід зазначити, що в C# клас в списку предків може мати лише один клас та декілька інтерфейсів перерахованих через кому.

Ще одна перевага інтерфейсів, це можливість за допомогою підтримки стандартних інтерфейсів середовища .Net користатися різноманітними методами стандартних об’єктів. Наприклад, щоб сортувати ваші об’єкти за допомогою класу **System.Array** – достатньо успадкувати їх від стандартного інтерфейсу **IComparable** і описати метод `int` **CompareTo**`(object obj)`. Цей метод повинен повертати:

- нуль – у випадку коли об’єкти, що порівнюються однакові;
- від’ємне ціле число – коли поточний об’єкт менше об’єкту-аргументу;
- додатне ціле число – коли поточний об’єкт більше об’єкту-аргументу.

Наприклад, нехай:

- одна крапка рівна іншій, коли обидві їх координати рівні;
- крапка **A(x1, y1)** більша крапки **B(x2, y2)**, коли перша координата **A** більша за першу координату **B(x1 > x2)**, або коли **x1=x2** і **y1>y2.**

Розглянемо відповідну програму:

```
using System;
public class Point : IComparable {
    double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public override string ToString() {
        return String.Format("Крапка({0},{1})", this.x, this.y);
    }
    public int CompareTo(object obj) { //метод від IComparable
        Point p = (Point)obj;
        if(this.x > p.x) {
            return 1;
        } else if(this.x < p.x) {
            return -1;
        } else {
            return (int)(this.y - p.y);
        }
    }
}
class SortablePoints {
    static void Main(string[] args) {
        Point p1 = new Point(5, 8);
        Point p2 = new Point(7, 12);
        Point[] points = new Point[5];
        points[0] = p1;
        points[1] = p2;
        points[2] = new Point(5, 10);
        points[3] = new Point(5, 1);
        points[4] = new Point(15, 10);
        // для сортування використовується
        // стандартний метод Sort
        Array.Sort(points);

        foreach(Point p in points) {
            Console.WriteLine(p);
        }
        Console.ReadKey();
    }
}
```

Вона виведе на екран крапки у відсортованому вигляді:

```
Крапка(5,1)
Крапка(5,8)
Крапка(5,10)
Крапка(7,12)
Крапка(15,10)
```

Щоб перевірити чи підтримує об’єкт той чи інший інтерфейс, можна скористатися одним з операторів **is** або **as**.

Оператор **is** – використовується для перевірки сумісності змінної або об’єкта одного типу з іншим типом даних і повертає значення **true** або **false**.

Наприклад:

```
int i;
if(i is IComparable){
    Console.WriteLine("Підтримує IComparable");
}
```

Дасть ствердну відповідь.

Оператор **as** – використовується для перетворення змінної одного типу в інший сумісний тип. Якщо типи не сумісні поверне **null**. Тому наступний фрагмент коду:

```
Point p0 = new Point(1, 2);
IConvertible c = p0 as IConvertible;
if(c == null) {
    Console.WriteLine("Не підтримує IComparable");
}
```

Видасть:

```
Не підтримує IComparable
```

9.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Навести власні приклади використання інтерфейсів.
3. Організувати сортування об’єктів за допомогою **IComparable**.
4. Перевірити роботу операторів **is** та **as**.
5. Налагодити програму на комп’ютері.
6. Забезпечити вивід результатів роботи програми на екран комп’ютера.
7. Підготувати звіт по роботі.

9.3. Контрольні запитання

1. Що таке інтерфейс?
2. Які переваги використання інтерфейсів?
3. Як оголошується інтерфейс?
4. Дайте означення операторів **is** та **as.**
5. Як використовуються оператори **is** та **as** для перевірки підтримки інтерфейсів?

10. Заняття №10.

Робота з рядками, датою та часом

Мета роботи – знайомство з класами .Net для роботи з рядками, датами та часом.

10.1. Теоретичні відомості

Рядки (**`String`**) – тип даних для зберігання текстової інформації. Тип даних **`String`** – це послідовність, що не містить жодного або довільну кількість символів Юнікоду в UTF-16, тобто по 2 байти на кожен символ. В C# ключове слово `string` є псевдонімом для `String`. Тому `string` і `String` – еквівалентні.

Не дивлячись на те, що `String` – об'єктним типом, оператори `==` та `!=` визначені для порівняння вмісту об'єктів типу рядок, а не посилань як для інших класів. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class WorkWithStrings {
    static void Main(string[] args) {
        string a = "hello";
        string b = "h";
        // Append to contents of 'b'
        b += "ello";
        Console.WriteLine(a == b);
        Console.WriteLine((object)a == (object)b);
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
True
False
```

Що показує, те що вміст рядків однаковий, хоча об'єкти різні.

Для рядків `String` визначено оператори `+` та `+=` для додавання. Проте слід зазначити, що ці об'єкти являються незмінними: після створення їх неможливо змінити. Все методи `String` і оператори C#, які, начебто змінюють рядок, насправді повертають в результаті новий об'єкт-рядок, що, іноді, може суттєво обмежувати швидкодію програмного коду (тому, іноді замість `String`, необхідно використовувати `StringBuilder`).

```
using System;
class WorkWithStrings {
    static void Main(string[] args) {
        string a, b = "h";
        a = b;          //a і b посилаються на один об'єкт
        Console.WriteLine((object)a == (object)b);
        b += "ello";    //тепер це різні об'єкти
        Console.WriteLine((object)a == (object)b);
        Console.WriteLine("a==" + a);
        Console.WriteLine("b==" + b);
        Console.WriteLine("a[0]: " + a[0]);
        Console.ReadKey();
    }
}
```

На екрані буде:

```
True
False
a==h
b==hello
a[0]: h
```

Тобто спочатку посилання `a` і `b` були на один об’єкт, а після виконання оператору `+=` змінна `b` стала вказувати на новий об’єкт.

Оператор `[]` служить тільки для доступу для читання окремих символів, та не може використовуватись для зміни значень.

Рядкові константи можна задавати в двох варіантах:

- в подвійних лапках;
- в подвійних лапках з `@` на початку.

В першому варіанті, використовуються Escape-послідовності, а в другому ні. Тому другий варіант зручний, щоб вказувати шлях до файлу.

Escape – послідовності:

Escape – послідовність	**Опис**
`\a`	Дзвоник ("alarm")
`\b`	Повернення на одну позицію ("backspace")
`\f`	Перехід на нову сторінку
`\n`	Перехід на новий рядок ("new line")
`\r`	Повернення каретки ("return")
`\t`	Горизонтальна табуляція ("tab")
`\v`	Вертикальна табуляція ("vertical tab")
`\0`	`null`
`\’`	Символ - ’
`\”`	Символ – ”
`\\`	Символ - \
`\udddd` або `\xdddd`	Представляє знак Юнікоду де dddd – шістнадцяткове число

Наприклад:

```
using System;
class WorkWithEscapes {
    static void Main(string[] args) {
        Console.WriteLine("\u0041BC\n\\");
        Console.WriteLine(@"C:\Temp\file1.txt");
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
ABC
\
C:\Temp\file1.txt
```

Основні властвості та поля класу **`String:`**

Назва	Опис
`Empty`	Статичне поле доступне тільки для читання, повертає пустий рядок
`Length`	Властивість, що повертає довжину рядка

Основні методи класу **`String:`**

Назва	Опис
`Compare`	Статичний метод для порівняння рядків
`Contains`	Перевіряє, чи міститься підрядок в рядку
`Copy`	Статичний метод. Створює новий об’єкт, що є копією даного рядка
`CopyTo`	Копіює заданий діапазон символів в з рядка в масив символів
`EndsWith`	Перевіряє, чи завершується рядок заданим підрядком
`Format`	Статичний метод. Дозволяє створити новий рядок зі значень різних виразів за допомогою форматування
`IndexOf`	Повертає індекс входження підрядку або набору символів в рядок
`IndexOfAny`	Повертає індекс входження будь-якого з набору символів в рядок
`Insert`	Вставляє підрядок в рядок
`IsNullOrEmpty`	Статичний метод. Перевіряє чи є рядок пустим або `null`
`Join`	Статичний метод. Створює рядок з масиву рядків використовуючи заданий роздільник
`LastIndexOf`	Повертає індекс останнього входження підрядку або набору символів в рядок
`LastIndexOfAny`	Повертає індекс останнього входження будь-якого з набору символів в рядок
`Remove`	Видаляє з рядка задану кількість символів
`Replace`	Замінює входження символу або підрядку новим символом або підрядком

Назва	Опис
`Split`	Створює з рядку масив підрядків за вказаним набором символів роздільників
`StartsWith`	Перевіряє, чи починається рядок з заданого підрядку
`Substring`	Створює новий рядок з підрядку поточного рядку
`ToCharArray`	Створює масив символів з підрядка
`ToLower`	Переводить рядок до нижнього регістру
`ToUpper`	Переводить рядок до верхнього регістру
`Trim`	Видаляє всі входження вказаних символів на початку та в кінці рядку
`TrimEnd`	Видаляє всі входження вказаних символів в кінці рядку
`TrimStart`	Видаляє всі входження вказаних символів на початку рядку

Якщо методів класу **`String`** не достатньо, для перетворень текстової інформації можна використовувати клас – **`StringBuilder`** (простір імен **`System.Text`**), що дозволяє змінювати окремі символи та має свій набір методів.

Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class WorkWithStringBuilder {
    static void Main(string[] args) {
        StringBuilder sb = new StringBuilder("kyiv");
        sb[0] = Char.ToUpper(sb[0]);
        Console.WriteLine(sb.ToString());
        Console.ReadKey();
    }
}
```

Ця програма замінює першу літеру слова `kyiv` на заголовну, і виводить на екран наступне:

```
Kyiv
```

Для роботи з датою та часом в середовищі .Net реалізовано структури **`DateTime`** та **`TimeSpan`.**

Структура **`DateTime`** представляє дату та час в діапазоні від 00:00:00 1 січня 0001 року (н. е.) до 23:59:59 31 грудня 9999 року (н. е.). Значення часу вимірюється в 100-наносекундних одиницях, що називаються тактами, і точна дата представляється числом тактів від 00:00:00 1 січня 0001 року (н.е.) по григоріанському календарю.

Основні поля структури **DateTime:**

Назва	Опис
MaxValue	Статичне. Повертає найменше можливе значення **DateTime**, доступне тільки для читання
MinValue	Статичне. Повертає найбільше можливе значення **DateTime**, доступне тільки для читання

Основні властивості структури **DateTime:**

Назва	Опис
Date	Для екземпляру **DateTime** повертає тільки дату без часу
Day	Для екземпляру **DateTime** повертає номер дня місяцю від 1 до 31
DayOfWeek	Для екземпляру **DateTime** повертає день тижня
DayOfYear	Для екземпляру **DateTime** повертає номер дня року від 1 до 366
Hour	Для екземпляру **DateTime** повертає години від 0 до 23
Millisecond	Для екземпляру **DateTime** повертає мілісекунди від 0 до 999
Minute	Для екземпляру **DateTime** повертає хвилини від 0 до 59
Month	Для екземпляру **DateTime** повертає місяць від 1 до 12
Now	Статичний. Повертає поточні дату та час.
Second	Для екземпляру **DateTime** повертає секунди від 1 до 59
Ticks	Для екземпляру **DateTime** повертає кількість тактів
TimeOfDay	Для екземпляру **DateTime** повертає час без дати
Today	Статичний. Повертає поточну дату
Year	Для екземпляру **DateTime** повертає рік від 1 до 9999

Основні методи структури **DateTime:**

Назва	Опис
Add	Додає до екземпляру **DateTime** значення типу **TimeSpan**
AddDays	Додає до екземпляру **DateTime** вказане число днів
AddHoura	Додає до екземпляру **DateTime** вказане число годин
AddMilliseconds	Додає до екземпляру **DateTime** вказане число мілісекунд
AddMinutes	Додає до екземпляру **DateTime** вказане число хвилин
AddMonths	Додає до екземпляру **DateTime** вказане число місяців
AddSeconds	Додає до екземпляру **DateTime** вказане число секунд
AddTicks	Додає до екземпляру **DateTime** вказане число тактів
AddYearS	Додає до екземпляру **DateTime** вказане число років
DaysInMonth	Статичний. Для вказаного місяця повертає кількість днів
Parse	Статичний. Перетворює рядок в **DateTime**, якщо це можливо, якщо ні генерує помилку
TryParse	Статичний. Перетворює рядок в **DateTime**, і повертає значення типу **Boolean** з інформацією про успішність перетворення

Структура **TimeSpan** – представляє інтервал часу, що може містити як додатню так і від'ємну кількість днів, годин, хвилин, секунд та долей секунд.

Основні поля структури **`TimeSpan`**:

Назва	Опис
`MaxValue`	Статичне. Повертає найменше можливе значення **`TimeSpan`**, доступне тільки для читання
`MinValue`	Статичне. Повертає найбільше можливе значення **`TimeSpan`**, доступне тільки для читання
`TicksPerDay`	Статичне. Константне. Кількість тактів в 1 дні
`TicksPerHour`	Статичне. Константне. Кількість тактів в 1 годині
`TicksPerMillisecond`	Статичне. Константне. Кількість тактів в 1 мілісекунді
`TicksPerMinute`	Статичне. Константне. Кількість тактів в 1 хвилині
`TicksPerSecond`	Статичне. Константне. Кількість тактів в 1 секунді
`Zero`	Статичне. Повертає нульове значення **`TimeSpan`**, доступне тільки для читання

Основні властивості структури **`TimeSpan`**:

Назва	Опис
`Days`	Повертає ціле число – кількість днів в поточній структурі **`TimeSpan`**
`Hours`	Повертає ціле число – кількість годин в поточній структурі **`TimeSpan`**
`Milliseconds`	Повертає ціле число – кількість мілісекунд в поточній структурі **`TimeSpan`**
`Minutes`	Повертає ціле число – кількість хвилин в поточній структурі **`TimeSpan`**
`Seconds`	Повертає ціле число – кількість секунд в поточній структурі **`TimeSpan`**
`Ticks`	Повертає значення поточного інтервалу **`TimeSpan`** в тактах
`TotalDays`	Повертає дійсне, що відповідає значенню поточного інтервалу **`TimeSpan`** в днях
`TotalHours`	Повертає дійсне, що відповідає значенню поточного інтервалу **`TimeSpan`** в годинах
`TotalMilliseconds`	Повертає дійсне, що відповідає значенню поточного інтервалу **`TimeSpan`** в мілісекундах
`TotalMinutes`	Повертає дійсне, що відповідає значенню поточного інтервалу **`TimeSpan`** в хвилинах
`TotalSeconds`	Повертає дійсне, що відповідає значенню поточного інтервалу **`TimeSpan`** в секундах

Розглянемо програму, що демонструє всі ці властивості.

```
using System;
class TimeSpanPropertiesDemo {

    const string dataFmt = "{0,-12}{1,8}       {2,-18}{3,21}";

    static void ShowProperties(TimeSpan interval) {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine(dataFmt, "Days", interval.Days,
            "TotalDays", interval.TotalDays);
        Console.WriteLine(dataFmt, "Hours", interval.Hours,
            "TotalHours", interval.TotalHours);
        Console.WriteLine(dataFmt, "Minutes", interval.Minutes,
            "TotalMinutes", interval.TotalMinutes);
        Console.WriteLine(dataFmt, "Seconds", interval.Seconds,
            "TotalSeconds", interval.TotalSeconds);
        Console.WriteLine(dataFmt, "Milliseconds",
            interval.Milliseconds, "TotalMilliseconds",
            interval.TotalMilliseconds);
        Console.WriteLine(dataFmt, null, null,
            "Ticks", interval.Ticks);
    }

    static void Main() {
        ShowProperties(new TimeSpan(10, 20, 30, 40, 50));
        Console.ReadKey();
    }
}
```

На екрані буде:

```
  10.20:30:40.0500000
Days            10      TotalDays              10,8546302083333
Hours           20      TotalHours                   260,511125
Minutes         30      TotalMinutes                 15630,6675
Seconds         40      TotalSeconds                  937840,05
Milliseconds    50      TotalMilliseconds             937840050
                        Ticks                  9378400500000
```

Для структур **DateTime** та **TimeSpan** визначено наступні оператори:

– Порівняння ==, !=, >, <, >=, <=, причому об’єкт типу **DateTime** порівнюють з об’єктом типу **DateTime**, а об’єкт типу **TimeSpan** – з об’єктом типу **TimeSpan;**

– Додавання: до об’єкту **DateTime** можна додати **TimeSpan** отримаємо значення типу **DateTime;** до **TimeSpan** додати **TimeSpan** отримаємо **TimeSpan;**

DateTime + TimeSpan -> DateTime
TimeSpan + TimeSpan -> TimeSpan

– Віднімання:

DateTime – DateTime -> TimeSpan
DateTime – TimeSpan -> DateTime
TimeSpan – TimeSpan -> TimeSpan

– Унарні оператори **+** та **– :**

```
+TimeSpan -> TimeSpan
–TimeSpan -> TimeSpan
```

Наприклад:

```
using System;
class TimeSpanOperators{

    static void Main() {

        DateTime dt, dt2;
        dt = DateTime.Now;
        dt2 = new DateTime(2010, 10, 27, 15, 30, 44);
        TimeSpan ts = dt - dt2;
        Console.WriteLine("dt={0} dt2={1} ts={2}", dt, dt2, ts);
        TimeSpan ts2 = new TimeSpan(1, 2, 3, 4);
        dt = dt - ts2;
        dt2 = dt2 + ts2;
        ts = ts + ts2;
        Console.WriteLine("dt={0} dt2={1} ts={2}", dt, dt2, ts);
        Console.ReadKey();
    }
}
```

Виведе на екран приблизно наступне:

```
dt=27.05.2010 18:52:30 dt2=27.10.2010 15:30:44
ts=-152.20:38:13.9843750
dt=26.05.2010 16:49:26 dt2=28.10.2010 17:33:48
ts=-151.18:35:09.9843750
```

10.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. За допомогою помічника (Help – клавіша F1) та бібліотеки MSDN ознайомитися з полями, властивостями та методами класів **String** та **StringBuilder,** структур **DateTime** та **TimeSpan.**
3. Виконати індивідуальне завдання за допомогою класу **String,** та окремо, те ж саме завдання за допомогою **StringBuilder**.
4. Перевірити роботу структур **DateTime** та **TimeSpan,** наприклад: визначити свій вік в днях, годинах, секундах і т.д., визначити час виконання програми, тощо.
5. Налагодити програму на комп’ютері.
6. Забезпечити вивід результатів роботи програми на екран комп’ютера.
7. Підготувати звіт по роботі.

10.3. Індивідуальні завдання

1. Дано рядок слів. Позбавитися лишніх пробілів та впорядкувати слова за зростанням.
2. Дано рядок чисел, що складається з чисел, розділених пробілами. Знайти суму цих чисел.
3. Дано повне ім’я файлу. Розкласти його на шлях, ім’я, розширення.
4. В тексті підрахувати середнє число слів у реченні.
5. В тексті знайти всі найдовші та найкоротші слова.
6. Підрахувати кількість входжень підрядку в рядок.
7. Дано текст, перевірити чи є він паліндромом (наприклад: «Де помити мопед?»).
8. Дано рядок, що містить круглі дужки. Перевірити правильність розташування дужок. У випадку помилки вказати позицію першої зайвої дужки.
9. Перетворити речення так, щоб усі слова починалися з великої літери.
10. Дано рядок, підрахувати кількість роздільників у рядку.
11. Дано рядок слів. Позбутися лишніх пробілів та вивести слова у зворотному порядку.
12. Дано рядок чисел. Позбутися лишніх пробілів, замінити пробіли на коми, та впорядкувати числа за зростанням.
13. Перевірити, чи являється введена з клавіатури строчка цілим числом, чі дробовим.
14. Написати програму, яка може бути перекладачем у «Королівстві кривих дзеркал»: слово, введене користувачем, переводиться на «задзеркальну» мову (літери в слові переставляються задом наперед). Програма повинна завершувати роботу після введення заданого символу, наприклад, зірочки.
15. Дано рядок символів, що складається з цифр, що відокремлені пробілами. Вивести на екран числа цього рядка в порядку зростання їх значень.
16. Дано рядок, що складається зі слів англійською мовою, які розділені пробілами. Вивести на екран ці слова в алфавітному порядку
17. Написати програму, яка перевіряє, чи є введений з клавіатури рядок двійковим числом.
18. З рядка, що складається з букв, цифр, ком, крапок, знаків "+" та "-", виділити підрядок, який відповідає запису цілого числа.
19. З рядка, що складається з букв, цифр, ком, крапок, знаків "+" та "-", виділити дійсне число з фіксованою крапкою.
20. З рядка, що складається з букв, цифр, ком, крапок, знаків "+" та "-", виділити дійсне число з плаваючою крапкою (наприклад, 1,2***10^-3**).

10.4. Контрольні запитання

1. Що таке рядки в середовищі .Net?
2. В чому відмінність між класами **`String`** та **`StringBuilder`**?
3. Назвіть основні методи та властивості класів **`String`** та **`StringBuilder`**?
4. Які структури використовуються для роботи з датами та часом в середовищі .Net?
5. В чому відмінність між структурами **`DateTime`** та **`TimeSpan`**?
6. Назвіть основні методи та властивості **`DateTime`**?
7. Які основні поля та властивості структури **`TimeSpan`**?
8. Які оператори визначені для роботи з структурами **`DateTime`** та **`TimeSpan`**?

Зміст звіту з лабораторної роботи

Звіт оформляється на аркушах формату А4 і починається з титульного аркуша (див. додаток).

На інших аркушах йде виклад за таким планом:

- мета роботи;
- основні теоретичні відомості по темі (користуючись, наприклад, MSDN);
- завдання, що вибране відповідно до варіанта;
- надрукована програма на мові C# і результати розрахунків;
- аналіз результатів розрахунків та висновки.

Додаток. Зразок титульної сторінки

Міністерство освіти, науки, молоді та спорту України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра автоматизації

енергосистем

Заняття №1

Знайомство з Visual Studio .NET. Типи даних. Ввід та вивід.

Виконав: студент групи ЕК-01
Петренко Д.А.

Перевірив: асистент Іваненко А.А.

Київ 2011

Список літератури

1. Библиотека MSDN [Электронный ресурс], *http://msdn.microsoft.com/ru-ru/library/default.aspx*
2. Standard ECMA-334. C# Language Specification, *http://www.ecma-international.org/publications/standards/Ecma-334.htm*
3. Джейсон Прайс. Майк Гандэрлой. Visual C# .NET. Полное руководство.: Пер. с англ. – К.: ВЕК+, СПб.: КОРОНА принт, К.:НТИ, М.: Энтроп, 2004. ISBN 966-7140-36-9.
4. Троэлсен Э. C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2007. ISBN – 978-5-318-00750-7.
5. Дж. Бишоп, Н. Хорспул. C# в кратком изложении. Пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2005. ISBN 5-94774-211-X.
6. М.В. Сухарев. Основы Delphi. Профессиональный подход. – СПб.: Наука и техника, 2004. ISBN 5-94387-129-2.
7. Дейтел, Х. и др. C#: Пер. с англ. СПб.: БХВ-Петербург, 2006. ISBN 5-94157-817-2.
8. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1,2 Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2002. ISBN 5-7502-0210-0, ISBN 5-7502-0220-8.
9. Лабор В.В. Си Шарп: Создание приложений для Windows. Мн.: Харвест, 2003. ISBN 985-13-1405-6.