КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Івохін Є.В. , Махно М.Ф., Піскунов О.Г.

Розробка додатків засобами мови програмування C#

Навч.-метод. посібник для проведення лабораторних робіт для студентів вищих навчальних закладів спеціальності «системний аналіз»

Навчально-методичний посібник
з напряму "Системний аналіз"
для студентів вищих навчальних закладів

КИЇВ 2021

УДК 519.8 ББК П56

Рецензенти:
к.ф.-м.н., доц. Ю.В. Бойко
к.т.н., доц. В.Р. Кулян

Затверджено до друку вченою радою факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка

Івохін, Є.В.; Махно, М.Ф.; Піскунов, О.Г.

П56 Розробка додатків засобами мови програмування C#: Навч.-метод. посібник для проведення лабораторних робіт для студентів вищих навчальних закладів спеціальності «системний аналіз» /Є.В.Івохін, М.Ф.Махно, О.Г.Піскунов. – К.: Видавничо-поліграфічний центр "Київський університет", 2021. – 100 с.

ISBN 966-642-048-1

Викладено основи програмування мовою C# для для студентів спеціальності «системний аналіз» і тих, хто продовжує вивчення .NET-технологій. Послідовно розглянуто основні концепції об'єктно-орієнтованого програмування, базові поняття мови C# та технологічні аспекти програмування на платформі .NET Framework.

Матеріал посібника спрямований на засвоєння принципів створення додатків мовою C# з використанням консольних та віконних інтерфейсів, що може бути використано при створенні власних програм на основі об'єктно-орієнтованого підходу із застосуванням сучасних методів програмування в .NET.

Для студентів вищих навчальних закладів, магістрів, аспірантів, викладачів, а також фахівців, які займаються розробкою та впровадженням програмних засобів на основі сучасних технологій програмування.

УДК 519.8
ББК

ЗМІСТ

ПЕРЕДМОВА

C# це об’ектно- і компонентно-орієнтована мова програмування. C# надає мовні конструкції для безпосередньої підтримки такої концепції роботи. Завдяки цьому C# підходить для створення та застосування програмних компонентів. З моменту свого створення мова C# збагатилася функціями для підтримки нових робочих навантажень і сучасними рекомендаціями по розробці програмного забезпечення.

Мова C#, розроблена компанією Microsoft, є однією з найпопулярніших сучасних мов програмування на ринку розробки програм в різних країнах. Мова C# застосовується для створення локальних програм для ПК, розробки складних веб-сервісів або мобільних додатків. Мова, яка з'явилася для реалізації власних потреб платформи Microsoft .NET, поступово стала дуже популярною.

Розробка мови почалася в 1998 році, а перша версія побачила світло в 2001. Групою розробників керував відомий в професійних кругах фахівець Андерс Хейлсберг. На даний час нові версії C# виходять порівняно часто, а поточні доопрацювання, виправлення багів і розширення бібліотек ведеться практично на постійній основі.

В результаті розробки з’явилася мова, яка є дуже гнучкою, потужною та універсальною. На C# пишуть практично все, що завгодно, від невеликих веб-додатків до могутніх програмних систем, об'єднуючих в собі веб-структури, додатки для десктопів і мобільних пристроїв. Все це стало можливим завдяки зручному C-подібному синтаксису, строгій структуризації, величезній кількості фреймворків і бібліотек (їх число досягає декількох сотень).

Багато хто вважає, що це окрема версія мови C, але насправді це не так. У C# дійсно є багато конструкцій, схожих на C і C++, але так само в нім можна знайти елементи Паскаля або Java. Це не розвиток лінійки C, а нова мова, створена з нуля.

Попит на програмістів C# не зашкалює, цю мову складно назвати дуже модною в 2020 році, але за її допомогою, як і раніше, можна розробляти все, що потрібне в 2020-му: від ігор і додатків до веб-сервисов.

Особливістю її використання є повне поєднання з екосистемою Microsoft. Це зрозуміло, бо C# — це проект Microsoft. Як наслідок, у C# все гаразд з підтримкою та з бібліотеками. Компанія Microsoft приділяє значну увагу підтримці мови розробки, а тому регулярно з'являються оновлення та доповнення, виправляються виявлені баги в компіляторі, розширюються бібліотеки. Розробники зацікавлені в популяризації інструменту і прикладають до цього масу зусиль.

У світі C# бібліотеки є практично для всього, у тому числі, і для роботи з популярними в сучасному світі інформаційних технологій нейромережами та машинним навчанням (ML.NET). Це означає, що знання мови та засобів розробки дозволить використовувати всі можливості нейронних мереж в додатках і об'єднувати їх за допомогою однієї і тієї ж

мови програмування. А оскільки C# — мультиплатформенна мова, то машинне навчання можна вбудувати практично в усі види програмних додатків.

C# дозволяє створювати ефективні веб-додатки. Розробники надають докладну і розгорнену документацію на своїх офіційних ресурсах. Крім того, відповіді практично на будь-які питання, пов'язані з роботою в C#, можна знайти в мережі Інтернет.

Існує безліч підручників, курсів для новачків і початківців, відеопідбірок і інших повчальних матеріалів. Цій посібник дозволить дати студентам основи розробки додатків на C#, познайомитися з особливостями реалізації об’єктно- і компонентно-орієнтованого підходу та підготуватися до професійного використання мови C#.

МОДУЛЬ 1: ВСТУП ДО МОВИ ПРОГРАМУВАННЯ C#

1.1 ЛАБОРАТОРНА РОБОТА: ОБЧИСЛЕННЯ ПРОСТИХ ЧИСЕЛ

Зміст заняття: Основні відомості про мову програмування C#.

Створення консольного додатку для обчислення всіх простих чисел до заданого за визначенням.

- Познайомитися з описом змінних вбудованих типів (int, bool).
- Познайомитися з читанням цілих чисел зі стандартного входу.
- Познайомитися з виразами, операторами - вираз і операторами циклу for і while.
- Познайомитися з можливостями форматування даних і висновком в стандартний вихід.

Корисна інформація при розробці програми для лабораторної роботи:

- тест програми повинен зберігається в файлі з розширенням .cs;
- компіляція з командного рядка MS Visual Studio виконується за допомогою команди csc /out:a.exe *.cs
- бажано підключити системний простір імен System (іменовану область видимості) для перегляду за замовчуванням за допомогою директиви

using System;

Ця директива дає можливість використовувати різні прості типи, математичні функції (Math.Sqrt - добування кореня) і стандартні файли вводу - виводу без додаткового кваліфікування (тобто, для опису цілої змінної можна писати 'mt a', а не 'System.Int32 a');

- програма складається з класів, хоча б один з яких містить статичний метод Main (точка входу в додаток)

```
class primeNumber
 { static int Main ()
   { // тіло методу
   }
 }
```

• Рядкове читання тексту зі стандартного вводу виконується наступним чином:

```
string a = Console.ReadLine ();  // скорочена форма
string b = Console.In.ReadLine (); // повна форма
```

• Запис в стандартний потік виводу (і стандартний потік виводу помилок) виконується оператором у вигляді:

```
Console.WriteLine ( "Hello,"); // скорочена форма
Console.Out.WriteLine ( "world"); // повна форма
Console.Error.WriteLine ( "world"); // повна форма
Console.WriteLine ( "this is {0} {1} years old", "Tom", 123);
                                        // вивід з специфікаторами місця виводу
```

• Приклад конвертації числа з рядка в двійковий вигляд:

```
int N = Int32.Parse ( "157");
```

1.2 ЛАБОРАТОРНА РОБОТА: РЕШЕТО ЕРАТОСФЕНА

Зміст заняття: Використання масивів у мові C#.

Обчислення всіх простих чисел до заданого алгоритмом Решето Ератосфена.

- Познайомитися з описом, відведенням пам’яті для масивів та використанням масивів.
- Познайомитися з типами для вимірювання часу (DateTime, TimeSpan).
- Порівняти час роботи алгоритмів за визначенням і Решето Ератосфена. Час виводити у секундах, тобто для 1 хвилини 1 секунди має бути виведено 61 секунда.
- В ускладненій версії програми потрібно не лише вивести невикреслені числа масиву (тобто, прості), а змістити прості числа до початку масиву і зменшити його довжину, так що б масив не містив нулі (викреслені значення).

Корисна інформація при розробці програми для лабораторної роботи:

• Одновимірний масив оголошується наступним чином:

```
int [] a = null;
```

• Пам'ять під масив відводиться оператором:

```
a = new int [20];
```

Звільняти пам'ять після використання непотрібно, цим займається збирач сміття. Властивість Length повертає довжину масиву.

• Зміна довжини масиву з 20 до 4:

```
Array.Resize (ref a, 4);
```

Статичний метод Array.Resize() створює новий масив заданого розміру, копіює в нього вміст старого масиву і повертає посилання на новий масив. Ключове слово ref вказує, що потрібно повернути посилання на нову пам'ять.

- Структура DateTime надає дату і час від нуля години 1 січня 0001 року до 24 години 31 грудня 9999 року. Властивість DateTime.Now повертає час і дату поточного моменту.
- Операція різниці між двома структурами DateTime повертає структуру TimeSpan. Остання призначена для зберігання інтервалів часу. У структурі DateTime є властивість (DateTime.DayOfWeek) яка надає назви дня тижня (наприклад, понеділок). У TimeSpan таких властивостей немає.
- Порівнювати час роботи двох додатків має сенс якщо вони працювали довше ніж по одній секунді.

1.3 ЛАБОРАТОРНА РОБОТА: ПЕРЕДАЧА ФАКТИЧНИХ ПАРАМЕТРІВ У МЕТОДИ

Зміст заняття: Вступ до мови програмування C#. Методи. Передача параметрів.

Створення консольного додатку для використання статичних методів.

- Вивчити три групи типів C# (Значущі типи, посилальні типи і рядки).
- Познайомитися зі способами застосування статичних і нестатичних методів.
- Познайомитися з правилами передачі фактичних параметрів в методи, і особливостями їх (параметрів) зміни.
- Познайомитися з описом своїх класів і структур, Створити новий клас і нову структуру для оголошення і передачі параметрів цього типу в метод.
- Створити статичні методи для зміни значень змінних з усіх трьох груп типів: цілого і структури, масиву і класу, рядки.
- Додаток має містити приклади перевантаження методів, виклику статичних методів зі свого класу (з методу Program.Main викликати метод Program.f); виклику статичного методу з чужого класу; виклику нестатичних методу.

Корисна інформація при розробці програми для лабораторної роботи.

Зміст лабораторної роботи полягає в тому, що

1. в методі Main для кожної з трьох груп типів (значимі, посилальні і рядки) оголосити чотири змінних, присвоїти їм значення (наприклад, для цілих - 1, 2, 3, 4) і передати двома способами (без застосування ключового слова ref або out та з ними) в статичний метод f. У методі двома способами (без використання конструктора і з ним) змінити ці значення. Потім перевірити, яке значення буде містити кожна з змінних після повернення з f.

```
using System;
 namespace ConsoleApplication1
 {
    class Program
}
```

```
    {
     static void Main(string[] args)
      {
      int a = 1, b = 2, c = 3, d = 4;
      Console.WriteLine("a/b/c/d before f : {0}/{1}/{2}/{3}",a, b, c, d);
      f(a, ref b, c, ref d);
      Console.WriteLine("a/b/c/d after f : {0}/{1}/{2}/{3}",a, b, c, d);
     }
    static void f ( int p1, ref int p2, int p3, ref int p4)
  {
    p1 = 10; p2 = 10;
    p3 = new int();
    p3 = 10;
    p4 = new int();
    p4 = 10;
    Console.WriteLine("p1/p2/p3/p4 inside f: {0}/{1}/{2}/{3}",p1,p2,p3, p4);
   }
  }
}
```

2. Поза області видимості класу Program, створити свою структуру:

```
 struct ss
{
  public //на відміну від C++ члени структур за замовчуванням закриті
   int p;
}
```

Перевантажити статичний метод f для зміни змінних типу ss

```
 static void f (ss p1, ref ss p2, ss p3, ss int p4)
 { p1.p = 10; p2.p = 10; p3 = new ss ();
   p3.p = 10;
   p4 = new ss ();
   p4.p = 10;
  Console.WriteLine ( "p1 / p2 / p3 / p4 inside f: {0} / {1} / {2} / {3}",
  p1.p, p2.p, p3.p, p4.p);
}
```

Повторити виконані дії для чотирьох змінних типу ss:

```
ss sa, sb, sc, sd; // пам'ять під структури вже отримана
sa.p = 1; // і був використаний конструктор за замовчуванням
sb.p = 2;
sc.p = 3; sd.p = 4;
```

В обох випадках додаток повинен вивести повідомлення на зразок:

```
 a/b/c/d before f: 1/2/3/4
 p1/p2/p3/p4 inside f: 10/10/10/10
 a/b/c/ d after f: 1/10/3/10
```

```
}
```

За результатом роботи видно, що після застосування методу f, у викликаючому коді (області видимості методу Main) значення поміняли тільки змінні, які передавалися за допомогою ключового слова ref (тобто за посиланням) незалежно від використання конструктора структури або цілого. Це означає, що для фактичних параметрів значимого типу в метод передаються значення, а, власне, сам метод працює з копією фактичного параметра, а не безпосередньо з ним.

3. Створити власний клас:

```
class cc { public int p;
```

При оголошенні змінних код потрібно поміняти:

```
cc a = new cc (), b = new cc (), c = new cc (), d = new cc ();
a.p = 1; // для посилальних типів потрібно явно застосувати конструктор
b.p = 2; // для захоплення пам'яті
c.p = 3;
c.p = 4;
```

У результаті програма має вивести на кшталт такого:

```
a/b/c/d before f: 1/2/3/4
p1/p2/p3/p4 inside f: 10/10/10/10
a/b/c/d after f: 10/10/з/ю
```

Значення змінної с не змінюється після застосування методу f, так як в метод була передана копія посилання на об'єкт, після виділення пам'яті в методі f старе посилання, природно, було втрачена, і всередині методу змінювалося значення нового об'єкта, який був втрачений при виході з методу f. У разі змінної d в метод f передавалося посилання на посилання на об'єкт.

Аналогічні дії повторити для масивів і рядків та пояснити результати. Треба пам'ятати, що об'єкти класу string не можна змінювати.

Обов'язково спробувати розташувати статичний і не статичний методи в класі cc і застосувати їх.

1.4 ЛАБОРАТОРНА РОБОТА: ОБРОБКА РЯДКІВ

Зміст заняття: Основи роботи з текстовими даними.

Використання методів класу string і обробка аргументів командного рядка.

- Познайомитися з інформаційним сервісом для розробників програмного забезпечення MSDN ([23]).
- Познайомитися з передачею аргументів командного рядка в C#.
- Познайомитися з типами для роботи зі текстами (char, string, StringBuilder).
- Познайомитися з таблицями кодування символів національних алфавітів (кирилиці) (1251, 866, UTF-16, UTF-8).
- Познайомитися з методами кодування символів національних алфавітів (кирилиці) (Encoding).

```
}
```

Корисна інформація при розробці програми для лабораторної роботи.

При виконанні лабораторної роботи на інформаційному сервісі для розробників програмного забезпечення MSDN потрібно переглянути опис класів Char і String. Вибрати шість методів для обробки рядків. В їх число обов'язково повинен входити метод Split - для поділу рядка на масив підрядків і методі Join - для з'єднання масиву рядків в один рядок. Крім цього, з'ясувати як за допомогою класу Encode можна явно задавати кодування зчитаного або виведеного файлу.

Вважається, що файли стандартного вводу - виводу мають кодування 866 сторінки (операційної системи MS DOS). Це означає, що стандартним додатком блокнот (notepad.exe) не вдасться побачити слово 'привіт', виведене в стандартний потік виводу в такий спосіб:

```
Console.WriteLine ( "привіт");
c: /tmp> a.exe > 1.txt
c: /tmp>notepad 1.txt
```

Далі, з урахуванням того, що текст в класах Char і String усередині програми зберігається в кодуванні UTF-16, необхідно пам'ятати, що при виведенні за допомогою методу Console.WriteLine () відбувається зміна кодування з UTF-16 на 866, а при введенні через метод Console.ReadLine () – з 866 на UTF-16.

Аргументи командного рядка передаються в додаток в масиві args точки входу в додаток (статичний метод Main)

```
static void Main (string [] args)
{
 // код тут
}
```

причому пропуски використовуються для розділення всього рядка на слова, які потрапляють в елементи згаданого масиву. Символи подвійних лапок або апострофа можна використовувати для того, щоб включати пропуски в елементи масив args.

Додаток має отримати один аргумент командного рядка і вивести в стандартний потік результат застосування вибраних методів, наприклад,

```
c:/ tmp>a.exe 1привіт:ВОЛКУ3
довжина: 15
символів цифри: 13
літери: привітВОЛКУ
в верхньому регістрі: 1ПРИВІТ:ВОЛКУ3
в нижньому регістрі: 1привіт:волку3
розподіл символом ':': '1привіт', 'волку', '3'
пропуски: 0
```

}

1.5 ЛАБОРАТОРНА РОБОТА: ВИБІР ДВОХ СТОВПЦІВ (на прикладі ГЕОГРАФІЧНОЇ ШИРОТИ ТА ДОВГОТИ) З ТЕКСТОВОГО ФАЙЛУ. ОБРОБКА ВИКЛЮЧНИХ СИТУАЦІЙ.

Зміст заняття: Використання методів конвертації текстових чисел в числа і навпаки (TryParse, Parse, ToString). Обробка виключних ситуацій.

- Навчитися видавати підказку по використанню у відповідь на ключі ’-?’ або ’/?’.

- Навчитися отримувати номери колонок для введення через аргументи команд-ного рядка з повною діагностикою введення. Наприклад: c:/tmp>a.exe c:/tmp>a.exe -ltt NN1 -lng NN2
де ключ ’-ltt’ задає номер колонки для широти, а ’-lng’ - для довготи. Цілі вводяться з використанням методу TryParse, тобто, без обробки виключних ситуацій.

- Використовувати необхідні методи класів char і string для обробки текстів і видачі діагностики про помилки, що виникають в процесі введення (номер рядка і позиція в рядку, в якому виникла помилка).

- Освоїти використання блоків try-catch-finally і вивод тексту помилки з класу Exception;

- Познайомитися зі специфікаторами місця виведення методів Console.WriteLine, Console.ReadLine, String.Format. Використовувати ці можливості для виведення градусів географічної широти і довготи із заданою точністю в 6 або 8 знаків після коми.

Корисна інформація при розробці програми для лабораторної роботи.

Додаток для вибору широти і довготи з тексту файлу. Використання:
a.exe [-?] [-help] [-v] -ltt N1 -lng N2 [-sep SEP] [-8]
де
-help: видача поточного екрану
-? : видача поточного екрану
-v: видача помилок у файлі і часу роботи програми
-sep SEP: задати роздільник (за замовчуванням - символ пропуску)
-ltt N1: N1 номер колонки з широтою
-lng N2: N2 номер колонки з довготою
-8: виводити 8 знаків після коми, інакше - 6.

Після видачі підказки рекомендується припиняти роботу програми з кодом повернення відмінним від 0. Тобто, робота методу Main повинна завершуватися оператором:

return 1;

або оператором:

Environment.Exit (1);

Другbq варіант призводить до негайного припинення роботи програми, навіть, якщо виконується звернення в методі, відмінному від Main.

Для введення всіх перерахованих ключів рекомендується завести набір змінних:

}

```
bool vFlag = false;
int lttNo = -1;
int lngNo = -1;
char sep = ' ';
bool oFlag = false;
int lineNo = 0; // номер поточного рядка
int errs = 0; // кількість помилок при аналізі файлу.
```

Обробку ключів командного рядка можна виконати таким чином:

```
for (int i = 0; i <args.Length: i ++)
 { if (args [i] == "-?")
    { // видати підказку Environment.Exit (1);
    }
else if (args [i] .ToLower () == "-help")
  { // видати шдказку Environment.Exit (1);
  }
else if (args [i] .ToLower () == "-v")
  vFlag = true;
else if (args [i] .ToLower () == "-8")
  oFlag = true;
else if (args [i] .ToLower () == "-sep")
  sep = setChar (++ i, args, "роздільник");
else if (args [i] .ToLower () == "-ltt")
  lttNo = setInt (++ i, args, "широти");
else if (args [i] .ToLower () == "-lng")
  lngNo = setInt (++ i, args, "довготи");
}
if (lttNo <0) {
  //не заданий стовпець для ширини Environment.Exit (2);
}
if (lngNo <0) {
  //не заданий стовпець для довготи Environment.Exit (2);
}
if (lttNo == lngNo) {
  // стовпці повинні бути різні Environment.Exit (2);
}
```

Приклад методу для отримання параметра аргументу командного рядка. Метод може використовуватися для ключів -sep (переробити під char), -ltt і -lng.

```
 static int setInt (int i, string [] args, string par)
  { int ret = 0; if (i <args.Length) {
     if (int.TryParse (args [i], out ret))
        ; // з і-го рядка прочитано ціле
      else {

}
```

```
      // значення для параметра par неправильного типу
      Environment.Exit (2);
     }
   }
    else {
     // не задано значення для параметра par
     Environment.Exit (2);
   }
  return ret;
 }
```

Приклад методу для отримання наступного рядка даних з файлу:

```
static string [] getRec (string line, char sep = '')
{ string [] flds = null;
  char [] seps;
  if (line! = null)
   { int lastP = line.IndexOf ( '#');
     if (lastP> = 0) // видалили коментар
     line = line.Substring (0, lastP);
     line = line.Trim (); // відрізати лідируючі і завершальні пропуски
     if (sep == ' ') {// якщо роздільник - пропуск додати табуляцію
       seps = new char [2];
       seps [0] = ;
       seps [1] = '\t';
       flds = line.Split (Seps, StringSplitOptions.RemoveEmptyEntries);
                                  // кілька пропусків трактуються як один
     }
      else {// якщо коми або крапка з комою, то
         seps = new char [1]; // кожен символ роздільник створює окреме
                                    // поле
       seps [0] = sep;
       flds = line.Split(Seps);
     }
   }
  return lds;
}
```

Функцію в незмінному вигляді можна використовувати для читання CSV файлів. Приклад циклу для розбирання текстового файлу і вибору значень з двох стовпців за допомогою наведеного методу getRec:

```
int errors = 0;  int lineno = 0;
double aa = 0.0;  double bb = 0.0;
string s;  string [] r;
string fld = "no";
r = getRec (s); // другий параметр - використовує значення за
                  // замовчуванням – пропуск
}
```

```
        try
         {
          if (r.Length> 0) {// якщо рядок не порожній
           fld = "first"; //то читаємо дані
           aa = double.Parse(r [1]);
           ld = "second";
           bb = double.Parse (r [2]);
                  //в специфікаторами місця виведення задана точність
        Console.WriteLine ( "{0: 0.000000} {1: 0.000000}", aa, bb);
        }
       }
     catch (Exception e) {// ловимо всі виключення
       if (vFlag) {
          Console.Error.WriteLine ( "Line / field: exception: {0} / {1}: {2} ",
           lineno , ld , e.Message);
           errors ++;
       }
    }
  }
          Console.Error.WriteLine ( "there was {0} errors", errors);
```

Кожен блок try повинен завершуватися або блоком catch, або блоком finally, або обома. Блоки catch бажано впорядкувати за операціями успадкування. Нащадки повинні йти вище батьківських класів. Так як клас Exception є батьком двох класів для обробки виняткових ситуацій, які будуть найчастіше зустрічатись при використанні цього коду, то блоки catch для FormatException і IndexOutOfRangeException треба розташувати вище:

```
        try
        {
         if (r.Length> 0)
           {// якщо рядок не порожній
            aa = double.Parse (r [1]);
            bb = double.Parse (r [2]);
           }
        }
      catch (FormatException e) {
      //в рядку не містилося число з плаваючою крапкою
      }
      catch (IndexOutOfRangeException e) {
      // вихід за межі масиву
      }
      catch (Exception e) {
      // ловимо всі інші виключення
      }

  }
```

Деякі нащадки класу Exception містять більш детальну інформацію про помилку, ніж батьківський клас, але обидва розглянуті вище обробники і FormatException, і IndexOutOfRangeException містять лише властивості предків (Message, Source, StackTrace, TargetSite, HResult).

Зауваження. Використання ключа /debug в рядку компіляції проекту, наприклад, c:/tmp>csc /debug /out:a.exe *.cs призведе до того, що повідомлення про необроблені виключні ситуації будуть містити номер рядка файлу, в якому виникла помилка. Ця інформація зберігається в файлі a.pdb, що знаходиться поруч з файлом a.exe.

1.6 ЗАВДАННЯ ДЛЯ МОДУЛЬНОЇ РОБОТИ

Розробити консольний додаток для

1. символьного складання двох цілих чисел довільної довжини.
2. символьного множення двох цілих чисел довільної довжини.
3. символьного ділення двох цілих чисел довільної довжини.
4. простого редагування текстового файлу (потрібно прибирати пропуски та табуляції між словами і знаками пунктуації, вставляти відсутній пробіл між знаком пунктуації та словом).
5. зміни кодування текстового файлу.

a.exe -866to1251 | -1251to866 | -1251toutf8

6. спіралевидной нумерації осередків двовимірного масиву (int [,] arr;)
7. зміни однієї з трьох форм представлення градусів (створивши свій клас для конвертації):
   - градуси з дробової частиною;
   - цілі градуси, хвилини з дробовою частиною;
   - цілі градуси, цілі хвилини, секунди з дробовою частиною.
8. обчислення відстані (або ступіні схожості) між двома окремими словами. Наприклад, між словами ’помилка’ і ’помилку’ відстань 1.
9. видалення однорядкових і багаторядкових коментарів з програм.
10. заміни макросів в тексті програми (мовою Сі) на їх значення.

МОДУЛЬ 2: ПРОГРАМУВАННЯ НА ПЛАТФОРМІ .Net FRAMEWORK

2.1 ЛАБОРАТОРНА РОБОТА: ВИКОРИСТАННЯ БІБЛІОТЕК, ЩО ПІДКЛЮЧАЮТЬСЯ ДИНАМІЧНО

Зміст заняття: Додатки, іменовані області видимості і динамічно підключені бібліотеки.

Розробити консольний додаток:

- або для демонстрації діалогового вікна,
- або для демонстрації використання бібліотеки обробки командного рядка [9].
- Познайомиться зі проектами (збірками) .Net.

}

- Познайомиться з просторами імен (областями видимості) і правилами використання типів з збірки.

- Підключити одну з запропонованих збірок до консольного додатку і продемонструвати використання типів цієї збірки.

Корисна інформація при розробці програми для лабораторної роботи.

- для підключення нової збірки, оформленої у вигляді бібліотеки, яка динамічно підключається при компіляції за допомогою командного рядка, потрібно використовувати ключ /r.

- для підключення до консольного додатку бібліотеки .Net що містить типи для створення віконних додатків, використовуйте командний рядок:

```
csc /r:System.Windows.Forms.dll /out:a.exe *.cs
```

При компіляції з цим ключем можна використовувати типи зі збірки, наприклад, Form (клас використовується для створення вікон на дисплеї) з зазначенням повної назви. Повна назва має вигляд: System.Windows.Forms.Form, де перші три слова визначають простір імен (або іменовану область видимості), а останнє Form - назву класу.

- для використання короткої форми імені типів з збірки в код програми необхідно додати директиву

```
using System.Windows.Forms;
```

Після цього клас Form та клас MessageBox (клас для підтримки передачі повідомлень) можна використовувати в повній та скороченій формах.

- клас MessageBox містить кілька перевантажених статичних методів Show для демонстрації різних видів діалогових вікон (з визначенням заголовка вікна, визначенням тексту в вікні, визначенням кількох кнопок, визначенням іконок). Крім того, методи повертають кнопку, натиснуту оператором. за допомогою перерахування DialogResult.

- для коректної роботи програми перед методом Main потрібно додати атрибут STAThreadAttribute:

```
[STAThread] static void Main ()  {}
```

Окрім цього, можна використовувати клас Chart для відображення деяких кривих у вікні типу Form.

Для виконання складнішої версії пропонується використовувати бібліотеку для розбору аргументів командного рядка [18]. Компіляція програми, як і у випадку системної бібліотеки, виконується наступним чином:

```
csc /r:args.dll /out:a.exe *.cs
```

Область видимості підключається за допомогою оператора

```
using Args;
```

У бібліотеці доступні кілька типів: ArgFlg - для введення логічних значень, ArgStr - для вводу рядків, ArgChar - для вводу символів, ArgInt - для вводу цілих, ArgFloat - для введення дійсних. Ці ж типи використовуються

```
}
```

для формування сторінки підказки додатку в методі Arg.mkVHelp. Кожен клас має метод для перевірки та вводу значення - check.

```
#pragma warning disable 642
using System;
using System.IO;
using Args;
class a{
  static void Main(string[] args) {
   ArgFloat    //   ключ типу float
      perCent = new ArgFloat(0.05, "p", "percent", "percent for something", "PPP");
  ArgStr      //    ключ типу рядок
      flNm   = new ArgStr("somefile.dat", "f", "file", "data file", "FLNM");
  ArgFlg //
      vFlag = new ArgFlg(false, "v", "verbose",  "to show additional information");
  ArgFlg //
        hFlag   = new ArgFlg(false, "?",  "help",   "to show usage page");
    for (int i = 0; i<args.Length; i++){
       if (hFlag.check(ref i, args)) {
        Arg.mkVHelp("to test command line arguments" // підказка по утилите ,
        “”, vFlag, Arg.mkArgs( hFlag ,vFlag ,perCent ,flNm));
         Environment.Exit(l); }
      else if (vFlag.check(ref i, args));
      else if (perCent.check(ref i, args));
      else if (lNm.check(ref i, args)):
   }
      Console.WriteLine("Ви задали: {0}/{1}/{2}/{3}" ,(bool) hFlag ,(bool) vFlag
        ,(double) perCent ,(string) lNm ); }
}
```

З ключем підказки '-?' додаток виведе наступне повідомлення:

```
to test command line arguments usage:
p [-?] [-v] [-p PPP] [-f FLNM] options:
-? : To show usage page: True
-v: to show additional information: False
-p PPP: percent for something: 0.05
-f FLNM: data file: somefile.dat
```

Для компіляції власних бібліотек, які підключаються динамічно, використовується ключ /t:

```
csc /t:library /out:l.dll *.cs
```

При цьому, бажано, щоб назва простору імен в бібліотеці (іменована область видимості) збігалася з ім'ям файлу. Хоча б один клас в бібліотеці повинен мати модифікатор доступу public:

```
namespace l { public class c {} }
```

}

Клас *c* у цьому випадку буде доступний в коді програми, що будується в такий спосіб:

csc /r:l.dll * .csc

2.2 ЛАБОРАТОРНА РОБОТА: СОРТУВАННЯ ТА КОЛЕКЦІЇ

Зміст заняття: Основи використання Framework .Net.

Розробити консольний додаток для сортування текстового CSV файлу (прізвище та вік студента) за прізвищем або за віком. Дані отримати з заданого в командному рядку файлу (з нестандартного введення).

- Познайомитися з класами File, StreamReader;
- Познайомитися зі списками.
- Познайомитися з шаблонами.
- Познайомитися з можливостями .Net для сортувань колекцій (інтерфейс IComporable).

Корисна інформація при розробці програми для лабораторної роботи.

Додатки для лабораторної роботи з точки зору обробки текстового файлу виконують ті ж самі дії, що і додаток з лабораторної роботи 1.5. Причому очевидно, що читання двох полів таблиці не є принциповим обмеженням. Читання будь-якої скінченої кількості полів у записі вимагає нескладних змін в коді, який використовує метод getRec. Власне, сам метод getRec, як і setInt можна перенести в поточний додаток. Головна відмінність полягає в тому, що для сортування даних потрібно всі дані прочитати в оперативну пам’ять (наприклад, в масив).

Використання додатку для сортування даних (прізвище та вік) про студентів, має наступний вигляд:

a.exe [-?] [-help] [-v] [-s NNN] [-sep SEP] -f FLNM

де

-help: видача поточної підказки

-? : видача поточного екрану

-v: видача помилок у файлі і часу роботи програми

-sep SEP: задати роздільник. За замовчуванням - символ пробілу

-s NNN: NNN номер поля для сортування (перше або друге)

-f FLNM: FLNM ім'я CSV файлу для читання

Нагадаємо, що квадратні дужки навколо аргументу командного рядка означають необов'язковість аргументу. Виходячи з цього, аргумент -f FLNM вважається обов'язковим, і додаток не повинен працювати без цього аргументу.

Далі, вимога сортувати файл не передбачає розробку методу для сортування, а передбачає використання можливостей .Net, призначених для цієї мети. Можливості для сортування та читання файлів пропонуються у вигляді інтерфейсів (тобто, наборів методів і властивостей без реалізації).

Далі буде використовуватися клас StreamReader, призначений для

```
}
```

читання текстових файлів, який вимагає явне звільнення ресурсів, пов’язаних з читанням файлу. У термінах мови Сі - файл потрібно відкрити, прочитати і закрити. Відкриття файлу виконується в конструкторі, читання рядку (як і в разі стандартного вводу) - методом ReadLine, закриття - звичним методом Close або його аналогом, прийнятим в .Net - Dispose. Клас StreamReader має реалізований метод Dispose, який дістався у спадок від інтерфейсу IDisposable. Таким чином, код для читання буде виглядати наступним чином:

```
string flNm;
// змінна для імені файлу, який треба отримати в результаті обробки ...
// аргументів командного рядка
string s; string [] flds;
StreamReader sr = new StreamReader (flNm);
for (lineno = 1; (s = sr.ReadLine ())! = null; lineno ++)
  { lds = getRec (s);
}
sr.Dispose ();
```

Для використання системного методу сортування, необхідно створити свій клас - спадкоємець інтерфейсу-шаблону IComparable <тип>:

```
using System.Collections.Generic;
class X: IComparable <X> {
public string nm; // ім'я студента
public int age; // вік студента
static public int sortOrder; //поле для задання порядку сортування
public // обов'язковий для реалізації метод для порівняння
        // поточного і іншого об'єкта
int CompareTo (X other) {
  if (sortOrder == 1)
    return nm.CompareTo (other.nm);
  else
    return age.CompareTo (other.age);
}
// конструктор
public X (string s, string i) {
  this.s = s;
  this.i = int.Parse (i);
}
}
```

Клас X успадковується від інтрейфейса - шаблону IComparable <X> і зобов'язаний мати реалізацію методу int CompareTo (X other), в якій порівняння двох об'єктів класу X в одному випадку (sortOrder==1) виконується за допомогою порівняння імєн студешів, в іншому (sortOrder! = 1) за допомогою порівняння їх віку. Таким чином функція сортування буде розставляти об'єкти чи в порядку, заданому іменами, або у порядку, заданому віком.

```
}
```

Об'єкти цього класу будемо накопичувати в списку - шаблоні типу List <X>:

List <X> recs = new List <X> (); X rec;

а у наведений вище цикл потрібно вставити рядки для створення об'єктів цього циклу з додаванням їх до списку:

rec = new X (flds [0], flds [1]); recs.Add (rec);

Після виходу з циклу залишається отримати зі списку масив і, якщо це задано аргументів командного рядка, його впорядкувати:

X [] recs2 = recs.ToArray (); // отримати масив

X.sortOrder = 1; // поле статичне, доступ до нього можна через тип

X.sortOrder = 2; //задати порядок сортування

Array.Sort (recs2);

В якості результату роботи необхідно вивести дані в стандартний потік виводу в новому порядку.

2.3 ЛАБОРАТОРНА РОБОТА: ЗАКРИТТЯ ВІКНА

Зміст заняття: Основи використання Framework .Net.

Розробити віконний додаток, який уточнює наміри користувача закрити вікно. Студенти з парними номерами залікових книжок показують вікно питання з кнопками Так - Ні, з непарними номерами залікових книжок показують вікно питання з кнопками Ок - Cancel.

- Познайомитися зі класом Form.
- Познайомитися зі способами демонстрації вікон.
- Навчитися оформлювати питання користувачу за допомогою стандартного вікна MessageBox і перерахування MessageBoxButtons.
- Навчитися отримувати відповідь користувача додатків за допомогою стандартного вікна MessageBox і перерахування DialogResult.
- Познайомитися з делегатами, подією FormClosing і обработчиками подій.

Корисна інформація при розробці програми для лабораторної роботи.

Потрібно використати підключення збірки System.Windows.Forms.dll. Побудувати клас спадкоємець для класу Form:

```
class w: Form
{ public w (string nm)
  { Text = nm; // задати заголовок вікна
  }
}
```

Вивести об'єкт класу w на екран монітора комп'ютера потрібно одним з трьох можливих методів (кращею реалізацією є експерименти з усіма трьома варіантами ShowDialog (), Show (), Application.Run() з поясненням різниці):

```
}
```

```
static int Main () {
 w a = new w ("dialog");
 DialogResult rc = a.ShowDialog ();
// Console.WriteLine ( "rc = {0}", rc);
}
```

Методом ShowDialog показують діалогове вікно, яке блокує доступ до інших вікон додатке, виконання зупиняється на методі ShowDialog (код чекає закриття вікна). Можна повернути результат діалогу, об'єкт *а* не буде зруйнований і його можна показати ще раз.

Методом Show показують дочірні вікна. Код додатку в операторі застосування методу Show не чекає закриття вікна, вікно після закриття можна показати ще раз (об єкт не руйнується).

```
static int Main () {
w a = new w ( "dialog");
w b = new w ( "1");
b.Show ();
a.ShowDialog ();
b.Text = "222222222";
b.Show ();
a.ShowDialog ();
}
```

За допомогою методу Application.Run() демонструють головне вікно програми. Код чекає закриття вікна, об'єкт руйнується. Зазвичай застосування методу Show відбувається при існуючому головному вікні програми.

```
static int Main () {
w a = new w ( "Main");
Application.Run (a);
Application.Run (a); // тут буде необроблена виключна ситуація
}
```

Для різних подій, які можуть відбуватися в зв'язку з роботою додатку пропонується великий набір змінних - делегатів, кожна з яких представляє собою список покажчиків на методи спеціального типу event delegate, які виконуються при настанні події. Наприклад, при спробі закрити вікно (натиснути хрестик в правому верхньому куті вікна) будуть викликатися методи зі змінною

```
event System.Windows.Forms.FormClosingEventHandler FormClosing;
```

де тип event FormClosingEventHandler - тип event delegate - обробник події закриття вікна. При цьому, для того, щоб метод (обробник) можна було внести в змінну - делегат (підписати на подію), він повинен мати правильний тип - делегат. У нашому випадку це event FormClosmgEventHandler. Метод оголошується наступним чином

```
}
```

```
void fc (object sender, FormClosingEventArgs a) {
Console.WriteLine ( "Form Closing of {0} is here", Text);
}
void fc2 (object sender, EventArgs a) {
Console.WriteLine ( "Form Closing2 of {0} is here", Text);
}
```

Підписування методу на подію відбувається так:

```
public w (string nm) {
 Text = nm;
 FormClosing + = fc;
 FormClosing + = fc2;
  //в список можна покласти декілька покажчиків на методи
}
```

У методі fc другий формальний параметр може мати тип FormClosing-EventArgs (спеціальний для події закриття вікна) або батьківський EventArgs (підходить до всіх подій). Перший випадок краще, тому що в змінній спеціалізованого тип FormClosingEventArgs існують додаткові поля.

Щоб мати можливість скасувати закриття вікна, можна написати такий обробник:

```
void fc3 (object sender, FormClosingEventArgs a) {
  if (e.CloseReason == CloseReason.UserClosing) {
     DialogResult resul = MessageBox.Show (
            "Точно закриваємо вікно?", "", MessageBoxButtons.YesNo);
   if (resul == DialogResult.Yes) {
    a.Cancel = false;
   }
   else
     a.Cancel = true;
   }
}
```

в якому з'ясовується причина закриття вікна (потрібно переконатися, що вікно закриває користувач CloseReason.UserClosing, вікно може закриватися з іншої причини, наприклад, повністю завершує роботу операційна система). Для цього показується діалогове вікно з кнопками Yes і No, обробляється результат натискання на кнопки і за потреб скасовується закриття вікна.

2.4 ЛАБОРАТОРНА РОБОТА: ДВІЙКОВІ (БІНАРНІ) ФАЙЛИ

Зміст заняття: Потоки введення - виведення даних.

Розробити консольний додаток, який перетворює таблицю з числами з текстового файлу в двійковий і навпаки, з можливістю сортування. Ускладнена реалізація лабораторної повинна надавати можливість формувати вигляд таблиці (таблиця з двох колонок цілих чисел, таблиця з

```
}
```

трьох колонок цілих чисел, таблиця з двох колонок дійсних чисел, таблиця з трьох колонок дійсних чисел). Бажаним є використання абстрактного класу як батька класу запису для всіх чотирьох пропонованих версій таблиці.

- Познайомитися зі класами File і Directory.
- Познайомитися з інтерфейсом IDisposable і третьою формою використання using.
- Познайомитися зі абстрактними класами TextReader, TextWriter.
- Познайомитися зі класами BinaryReadrer, BinaryWriter.
- Познайомитися зі шаблонами списків.

Корисна інформація при розробці програми для лабораторної роботи.

Для використання додатку для конвертації текстових записів в двійкові і навпаки застосувати такий виклик:

a.exe [-?] [-v] [-s NNN] [-sep SEP] [-3] [-doule] {-txt2bm | -bm2txt} -f FLNM де

-help: видача поточної підказки
-? : видача поточного екрану
-v: видача помилок у файлі і часу роботи програми
-sep SEP: задати роздільник. За замовчуванням - символ пробілу
-txt2bm: напрямок конвертації - зі стандартного вводу в двійковий
-bm2txt: напрямок конвертації - з двійкового в стандартний вивід
-3: записи складаються з трьох чисел (за замовчуванням з двох)
-doudle: записи складаються з дійсних чисел (за замовчуванням з цілих)
-s NNN: NNN номер поля для сортування
-f FLNM: FLNM ім'я файлу, що обробляється

Вертикальна риса '|' означає вибір однієї з альтернатив, а фігурні дужки - означають обов'язковість одного або іншого ключа. Квадратні дужки, як і раніше, [-txt2bm | -bm2txt] означа.nm, що можна пропустити обидва ключа.

З лабораторної роботи 1.5 можна використати метод getRec для читання записів з текстового файлу, який написаний з урахуванням обробки коментарів в CSV-файлі, а в лабораторній роботі 2.2 обговорювалися питання сортування колекцій.

Для читання - запису простих типів в двійковому вигляді призначені класи BinaryReader і BinaryWriter. Але конструктори класів BinaryReader і BinaryWriter в якості параметрів не приймають назву файлу, як це було у StreamReader і StreamWriter. Вони приймають клас Stream, призначений для низькорівневого читання - запису. Об'єкти класу Stream можна отримати за допомогою методів класу File, наприклад, методом Open. Крім того, нагадаємо, що ці класи використовують важливий ресурс (файл файлової системи), тому вони є спадкоємцями інтерфейсу IDisposable і, отже, мають метод Dispose для звільнення свого ресурсу. Для зручності використання класів - спадкоємців інтерфейсу IDisposable в .Net існує спеціальний оператор using, який гарантує виклик методу Dispose незалежно від виникнення або невиникнення виключних ситуацій під час роботи з файлом.

}

Таким чином, при використанні інструкції using код для вводу даних буде виглядати так:

```
string flNm = "1.bin"; // ім'я файлу
using (BinaryReader br = new BinaryReader(File.Open(flNm, FileMode.Open)))
    { …..
    }
```

а код для виводу:

```
    using (BinaryWriter bw = new BinaryWriter (
    File.Open (lNm, FileMode.Create))
    { …..
    }
```

Перерахування FileMode задає різні режими для роботи з файлами:
- FileMode.Open - операційна система повинна відкрити існуючий файл. Відсутність файлу призведе до виключної ситуації FileNotFoundException. Застосування методу Write призведе до встановлення запису даних з початку файлу, таким чином, деякі зі попередніх даних будуть втрачені;
- FileMode.Create - операційна система створить новий файл, якщо його не було, якщо ж він був, файл буде перезаписаний;
- FileMode.Append - операційна система повинна відкрити існуючий файл; при цьому потік Stream позиціонується в кінець файлу. Застосування методу Write призведе до дописування даних в кінець файлу. Всі попередні дані залишаються.

Як вже згадувалося вище, в кожному з класів для вводу - виводу двійкових даних існує багато методів. Розглянемо деякі з них
для вводу даних:
- логічна змінна bool p = br.ReadBoolean ();
- ціла змінна int p = br.ReadInt32 ();
- коротке ціле short p = br.ReadInt16 ();
- довге ціле long p = br.ReadInt64 ();
- окремий байт byte p = br.ReadByte ();
- плаваюче одинарної точності float p = br.ReadSingle ();
- плаваюче подвійної точності double p = br.ReadDouble ();

і т.д.
для виводу (застосовється перевантажений метод Write):
- bw.Write (1); // ціле
- bw.Write (1L); // довге ціле
- bw.Write ((short) 1); // короткий
- bw.Write (2.71828F); // плаваюче одинарної точності
- bw.Write (2.718281828); // плаваюче подвійної точності
- bw.Write (true); // логічне
- bw.Write ("farewell companions, good luck, hurrah");
- bw.Write ("the boiling sea under us");

Конструкцію using можна використовувати в формі без конструктора, наприклад, зі стандартним вводом - виводом наступним чином:

```
}
```

```
using (TextWriter tw = Console.Out) {
using (TextReader tr = Console.In) {
//
}
}
```

де TextWriter і TextReader - це абстрактні класи, призначені для текстового вводу-виводу. Ці класи успадковуються класами StreamWriter і StreamReader, відповідно, які зазвичай використовуються при явному відкритті тестових файлів.

При виконанні лабораторної можна наслідувати типи BmaryReader і BinaryWriter, додавши до них необхідні методи для вводу-виводу двох цілих, трьох цілих, двох плаваючих і трьох плаваючих чисел.

2.5 ЛАБОРАТОРНА РОБОТА: ВИВЕДЕННЯ ТАБЛИЦІ У XML ФОРМАТІ

Зміст заняття: XML документи, серіалізація і десеріалізацію.

Розробити консольний додаток, який перетворює таблицю з CSV-файлу в XML - файл. Роздільником полів буде вважатися символ крапки з комою - ';', два роздільник не рахуються одним (як у випадку пропусків), а задають порожнє поле. Вхідний CSV-файл може мати різну кодування, вихідний XML - файл мати кодування UTF-8.

- Познайомитися с простором імен (областю видимості) System.Xml;
- Познайомитися с XML форматом.
- Номер рядка в початковому CSV - файлі виводити в якості атрибута відповідного запису в XML - файлі.
- Познайомитися з шаблонами словників, передбачити можливість явного іменування одного або більше полів у записі;
- Познайомитися з кодуваннями .Net. Використовувати для введення наступні кодування: 866, 1251, UTF-8;
- Підготувати кілька тестів для демонстрації працездатності додатки у вигляді командного файлу.

Корисна інформація при розробці програми для лабораторної роботи.

Формат використання додатку може мати приблизно такий вигляд:

a.exe [-?] -o PATH [-e ENC] [-s CHAR] [-f FLDNM1 [-f FLDNM2] ...]

де

-? : надання допомоги

-o : шлях для збереження файлу xml

-f FLDNMx: встановлення спеціального поля (номер: ім'я). За замовчанням: field

-e ENC: стандарт вхідного кодування (866, 1251, UTF-8) (за замовчанням 866)

-s CHAR: розхдільник («,»)

Текст [-f FLDNM1 [-f FLDNM2] ...] повідомляє оператору, що значення, які задаються ключем -f, будуть накопичуватися. Їх треба ввести

```
}
```

стільки, скільки полів передбачається явно іменувати. При цьому, значення з наступного (більш правого) ключа завжди замінюють значення з попереднього (лівого) ключа.

Для задання кодування стандартного потоку вводу використовується властивість InputEncoding класу Console, яка має тип Encoding. Об'єкти класу Encoding створюються з коротких або цілочисельних значень за допомогою статичних перевантажених методів цього ж класу - GetEncoding. Список кодувань, підтримуваних .Net, можна подивитися на сторінці https: //docs.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.8. Для поточної лабораторної представляють інтерес два вузькі (один байт - один символ) формати кодування та одне широке (один або два байти відповідають одному символу): 866 - використовувалася в DOS-і та в консолі; 1251 - використовувалася в Windows; UTF-8 - кодування, яке дозволяє користуватися кількома національними алфавітами.

Задати формат кодування стандартного вводу можна наступним чином:

```
Console.InputEncoding = Encoding.GetEncoding ( "cp866");
// Console.InputEncoding = Encoding.GetEncoding ( "windows-1251");
// Console.InputEncoding = Encoding.GetEncoding ( "utf-8");
```

Після виконання однієї з інструкцій метод ReadLine в операторі

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

буде виконувати перекодування тексту у внутрішню систему кодування UTF-16.

Для завдання імен полів може використовуватися шаблон словника:

```
using System.Collections.Generic;
Dictionary <int, string> fldDict;
```

Тип словника означає, що цілі числа будуть виконувати роль індексу (key в контексті словника), а рядки - значення. При цьому, якщо пара з таким індексом і значенням була раніше додана до словника, застосування операції індексування призводить до виключної ситуації

KeyNotFoundException: The given key was not present in the dictionary.

Додавання пари до словника проводиться таким чином:

```
static void addNm (string field, Dictionary <int, string> fldDict) {
string [] val = field.Split (':'); // розділити параметр аргументу ком.строки
try {
  int k = int.Parse (val [0]);
  if (fldDict.ContainsKey (k)) return;
  fldDict.Add (k, val [1]); // додати наступну пару ключ - значення
}
catch {
  Console.Error.WriteLine ( "wrong parameter: {0}", field);
}
}
}
```

Після обробки всіх ключів в якості імені поля з номером *i* будемо використовувати fldDict[i], якщо індекс *i* існує. Для цього використовується метод ContainsKey:

```
string nm = null;
if (fldDict! = null && fldDict.ContainsKey (i)) nm = ldDict [i];
```

У іншому випадку, в якості імені поля можна взяти рядок, що утворюється, наприклад, так: nm = string.Format ("field_ {0}", i);

Для виводу xml файлу потрібно задати область видимості System.Xml, і типи типи даних з неї: XmlDocument - документ і XmlNode - вузол. Документ складається з вкладених один в одного вузлів. Вузол можна вкладати в вузол в якості вузла або в якості атрибута. В атрибути не можна вкладати вузли.

Створити та зберегти xml-документ можна наступним чином:

```
XmlDocument xd = new XmlDocument ();
xd.LoadXml ( "<?xml version =\" 1.0 \"encoding =\" utf-8 \"?> \n <Table> </Table>");
XmlNode tbl = xd.DocumentElement;//перший вузол в документі- таблиця
xd.Save ( "doc.xml");
```

До отриманої таблиці tbl потрібно додавати записи:

```
XmlNode attr;
XmlNode rec;
int recNo = 0;
string str = null;
string [] vals;
  while ((str = tr.ReadLine ())! = null) {
    vals = getRec (str, ';');
    if (values.Length == 1 && values[0].Length <= 0) continue;
    recNo ++;
    rec = xd.CreateElement ( "Record"); // створили новий запис
    attr = xd.CreateNode (// створили атрибут - її номер
          XmlNodeType.Attribute, "No", null);
    attr.Value = recNo.ToString ();
    rec.Attributes.SetNamedltem (attr); // додали атрибут в запис
     ... // тут можна створити декілька полів.
  tbl.AppendChild (rec); // додали запис в таблицю
  }
```

Залишилося створити вміст запису, тобто, додати до вузла rec запис кількох вузлів, які будуть зберігатися в змінних fld. У кожного вузла fld повинна бути назва (яка має бути знайдена в словнику і збережена у раніше описаній змінній nm) та значення.

```
XmlNode fld;
for (int i = 0; i <values.Length; i ++) {
fld = xd.CreateElement (nm); // створили новий вузол - чергове поле
}
```

```
fld.InnerText = values// присвоїли йому значення
rec.AppendChild (fld); // додали його до запису
}
```

На цьому вивод в xml - файл завершено. Аргументи командного рядка дозволяють використовувати командні файли консолі для тестування додатків.

2.6 ЗАВДАННЯ ДЛЯ МОДУЛЬНОЇ РОБОТИ

Розробити консольний додаток

1. для обчислення проекції Меркатора, (див. [1, с.58] та методи, для відображення градусів сфери в пікселі зображення і, навпаки, пікселів в градуси).
2. для виведення координат руху транспортного засобу у вигляді XML - файлу.
3. для роботи програми, що стежить за станом каси магазину на основі двійкового файлу.
4. для роботи програми, що стежить за станом каси магазину на основі текстового файлу.
5. для виконання машини Тьюринга ([3]).
6. для транслітерації імен файлів на жорсткому диску.
7. для включення-відключення автозапуску для змінних накопичувачів (USB-флеш-накопичувачів, оптичних дисків (CD-ROM, DVD-ROM)).
8. для використання парсера математичних виразів ([22]).
9. для видачі решти. Через стандартний потік вводу додаток отримує стан каси у вигляді несортованого списку пар (Номінал_монети_або_банкноти, кількість_монет) по одній парі в рядку, через аргументи командного рядка - загальну суму решти. Додаток підбирає монети для заданої суми, і, додатково, виводить відсортований стан каси (по ключу -v).
10. утиліту для обчислення різниці в двох текстових файлах.

Розробити публічну збірку (бібліотеку, що динамічно підключається) з Суворими Іменами (Див. https://professorweb.ru/index.php).

ПИТАННЯ ДЛЯ ДИФЕРЕНЦІАЛЬНОГО ЗАЛІКУ

Тестові питання

1. Чому дорівнює змінна i? int i = 0; int [,] a = new int [3,4]; i = a.Length;
2. Чому дорівнює змінна i? int i = 0; int [,] a = new int [3,4]; i = a.Rank;
3. Чому дорівнює змінна i? int i = 0; int [,] a = new int [3,4]; i = a.GetLength(0);
4. Чому дорівнює змінна i? int i = 0; int [,] a = new int [3,4]; i = a.GetLength(1);
5. Чому дорівнює змінна i? int b = 2; int i = 0; i = b++;

```
}
```

6. Чому дорівнює змінна i? int b = 2; int i = 0; i = ++b;
7. Чому дорівнює змінна i? int b = 2; int i = 0; i = b + b++;
8. Чому дорівнює змінна i? int b = 2; int i = 0; i = b++ + b;
9. Чому дорівнює змінна i? int b = 2; int i = 0; i += b;
10. Чому дорівнює змінна i? int b = 2; int i = 0; i -= b;
11. Чому дорівнює змінна i? int i = 0; i = 1/2;
12. Чому дорівнює змінна i? int i = 0; i = 1
13. Чому дорівнює змінна f? double f = 0.0; f = -1.0; f = Math.Abs(f);
14. Чому дорівнює змінна f? double f = 0.0; f = -3.14159; f = Math.Round(f);
15. Чому дорівнює змінна f? double f = -2.0; f = Math.Pow(f, 2);
16. Чому дорівнює змінна f? double f = 0.0; f = Math.PI;
17. Чому дорівнює змінна f? double f = 4.0; f = Math.Sqrt(f);
18. Чому дорівнює змінна f? double f = 4.0; f = Math.Sin(0.0);
19. Чому дорівнює змінна f? double f = 4.0; f = Math.Cos(0.0);
20. Чому дорівнює змінна f? double f = 0.0; f = true?2.0:0.0;
21. Чому дорівнює змінна f? double f = 0.0; f = false?2.0:0.0;
22. Чому дорівнює змінна f? double f = 0.0; int z[]=null; f = z==null?2.0:0.0;
23. Чому дорівнює змінна f? double f = 0.0; int z[]=new int[2]; f = z==null?2.0:0.0;
24. Чому дорівнює змінна f? double f = 0.0; f = double.Parse("3");
25. Чому дорівнює змінна f? double f = 0.0; f = double.Parse(" 3 + 2");
26. Чому дорівнює змінна f? double f = 0.0; f = 1.0 + 2.0 * 3;
27. Чому дорівнює змінна f? double f = 0.0; f = (1.0 + 2.0) * 3;
28. Чому дорівнює змінна i? int b = 2; int i = 0; i = b++ + b++ + b++;
29. Чому дорівнює змінна i? int b = 2; int i = 0; i = ++b + ++b + ++b;
30. Приведіть приклади основних констант для вбудованих типів.

Контрольні питання

1. Особливості процесу побудови програми для .Net.
2. Що є результатом збирання проекту в .Net? Перерахуйте основні типи файлів MS Visual Studio для побудови програми.
3. Перерахуйте області видимості.
4. З чого складається програма на мові C#? Охарактеризуйте мову C#.
5. Перерахуйте і охарактеризуйте вбудовані типи мови C#. На які групи діляться всі типи C#?
6. Перерахуйте і охарактеризуйте оператори мови C#.
7. Перерахуйте і охарактеризуйте інструкції мови C#.
8. Наведіть приклади інструкцій оголошень для основних вбудованих типів.
9. Перерахуйте і охарактеризуйте інструкції вибору (розгалуження).
10. Перерахуйте і охарактеризуйте інструкції ітерації (циклу).
11. Перерахуйте і охарактеризуйте інструкції безумовного переходу.
12. Перерахуйте і охарактеризуйте ключові слова, що використовують-

}

ся для обробки виняткових (виключних) ситуацій.

13. Що таке клас і структура. Відмінності між класами і структурами. З чого складається клас або структура?

14. Статичні і нестатичні члени класів або структур.

15. Як виконується передача фактичних параметрів в методи? Ключові слова ref і out.

16. Який клас в .Net надає математичні функції. Приклади функцій.

17. Які структури в C# використовуються для роботи з часом?

18. Методи і властивості, які надає клас string?

19. Методи і властивості, які надає структура char?

20. Які escape-послідовності символів доступні в C# ?

21. Які методи, що не призводять і призводять до виключних ситуацій, використовуються для перетворення рядків в числові типи? І назад, з числових типів у рядки?

22. Які типи і методи використовуються для введення - виведення текстових файлів?

23. Які типи і методи використовуються для введення - виведення двійкових файлів?

24. Що таке метод класу? Приклади перевизначення методів класу.

25. Що таке конструктор класу? Як викликається конструктор батьківського класу. Приклади перевантажень конструкторів класу. Конструктор за замовчуванням.

26. Що таке властивість класу?

27. Що таке індексатор класу?

28. Що таке колекція в .Net? Навести два приклади колекції, і їх основні методи і властивості.

29. Що таке інструкція using?

30. Масиви в мові C#, ініціалізація масивів, одномірні і багатовимірні масиви. Основні властивості і методи масивів.

Додаткові питання

1. З яких розділів складається результат збирання проекту?

2. Основні директиви препроцесора C#. Для чого вони використовуються?

3. Що таке абстрактний клас?

4. Наведіть і охарактеризуйте приклади декількох типів, які використовуються для обробки виняткових станів.

5. Реалізуйте метод CompareTo в наступному класі class S: IComparable { bool first; string fio; uint age; } для сортування колекцій або по полю fio, або по полю age.

6. Додайте в клас class S { string fio; uint age; } конструктор і статичну властивість, для підрахунку кількості створених об’єктів.

7. Що таке перевантаження операторів? Охарактеризуйте перевантаження операторів в C#.

8. Для класу class coor { double ltt; double lng; } перевантажте

}

оператор перетворення у рядок.

9. Який механізм використовується для реалізації поліморфізму?

10. Пізнє і раннє зв'язування. Віртуальні функції і перевизначення функцій. Ключові слова virtual і override.

11. Що таке атрибути (System.Attribute)? Наведіть приклади часто використовуємих атрибутів.

12. Як з типу перерахування (Enum) отримати текстові подання всіх його значень і рядок перетворити в значення з типу перерахування?

13. Який клас в .Net використовується для створення вікон? які методи в .Net використовуються для демонстрації вікон оператору? Типи вікон.

14. Який клас в .Net використовується для створення керуючих елементів вікон? Яким способом керуючі елементи зв'язуються з вікном?

15. Що таке файлова система? Які класи використовуються для управлінням файлової системи (Створення, копіювання, видалення файлів і директорій)?

16. Ключове слово yield і приклад його використання.

17. Для чого використовується клас File.

18. Отримання випадкових чисел в .Net.

19. Що таке інтерфейс?

20. Для чого використовується клас Directory?

21. Для чого використовується клас DirectoryInfo?

22. Що таке автоматично реалізовані властивості (Auto-implemented properties)? Наведіть приклад.

23. Що таке клас StringBuilder?

24. Що таке лямбда - вираз? Приклади використання.

25. Які типи і методи використовуються для низькорівневого вводу - виводу файлів?

26. Що таке регулярний вираз?

27. Привести приклад і пояснити використання ключових слів checked, unchecked.

28. Що таке динамічний тип (dynamic)? Наведіть відмінності від типу object і типу var.

29. Приклади інтерфейсів в C#.

30. Що таке клас CultureInfo і як вона впливає на роботу програми?

МОДУЛЬ 3: СТВОРЕННЯ ВІКОННИХ ДОДАТКІВ

3.1 ЛАБОРАТОРНА РОБОТА: КЕРУЮЧІ ЕЛЕМЕНТИ

Зміст заняття: Віконні додатки.

Створити додаток для демонстрації заданого керуючого елемента зі списку (Button, TextBox, Label) в заданому місці, з заданим текстом

- Познайомитися зі можливостями .Net по динамічному створенню вікон.

- Познайомитися з класами TextBox, Label, Button, Control.

```
}
```

- В ускладненій версії програми потрібно продемонструвати відновлення стандартних налаштувань у вікні TextBox. Мається на увазі, що після закриття вікна і подальшої демонстрації його користувачу, тест в полі введення повинен стати вихідним. При цьому, новий об’єкт вікна не створюється, а використовується створений на початку.

Корисна інформація при розробці програми для лабораторної роботи.

Завершений приклад конструктора вікна має вигляд

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace m3l1{
class wnd : Form {
public
wnd(int x, int y, int f, string w) {
this.Width = 300; // зміна розмірів вікна
this.Height = 200;
Control cnt;
if (f == 1)
cnt = new Button();
else if (f == 2)
cnt = new Label();
else if (f == 3)
cnt = new TextBox();
else
return;
// cnt.Location = new Point(x, y);
cnt.Left = x; // зміна координат
cnt.Top = y; // лівого верхнього кута
cnt.Text = w; // керуючого елементу
Controls.Add(cnt);
}
  }
}
```

3.2 ЛАБОРАТОРНА РОБОТА: ДІАЛОГОВЕ ВІКНО OkCancel

Зміст заняття: Створення модальних вікон.

Розробити клас для демонстрації питання оператору з двома кнопками Так - Ні або Ok - Cancel і додаток для демонстрації вікна цього класу.

- Познайомитися з методами демонстрації вікон.
- Познайомитися з класом Form і властивостями вікна.
- Познайомитися з класами Button, Panel і можливостями прив’язки управлющіх елементів до вікна.

Корисна інформація при розробці програми для лабораторної роботи.

```
  }
```

Створюється батьківський клас ОкCancel для діалогового вікна з кнопками Ок та Cancel. Треба пам’ятати, що керуючий елемент з'являється у вікні після додавання елемента його до колекції Controls. Наприклад,

Controls.Add (butOk);

Крім цього потрібно налаштовувати деякі властивості, звичайні для діалогових вікон.

• що б мати можливість використовувати тип Size, у додаток треба додати бібліотеку System.Drawing.dll і в код додати рядок

using System.Drawing;

• створити клас ОкCancel, який успадковується з класу Form. Клас повинен містити дві кнопки

Button butOk;
Button butCancel;

• заборонити оператору змінювати розмір діалогового вікна і прибрати всі зайві елементи керування

FormBorderStyle = FormBorderStyle.FixedDialog;
MinimizeBox = false;
MaximizeBox = false;
ControlBox = false;
AutoScroll = false;

• створити змінну p = 4. Це число буде використовуватися для створення полів між границею вікна і його керуючими елементами. Властивість вікна називається Padding

Padding = new Padding (p, p, p, p);

• задати цю властивість вікна для того, щоб в ньому розміщувалися додаткові керуючі елементи

AutoSize = true;

• створити першу панель із заданою висотою і розмістити зверху на формі

Panel p0 = new Panel (); // ця панель задає висоту рядка
p0.Size = new Size (1, 32);
p0.Dock = DockStyle.Top;
Controls.Add (p0);

• створити кнопку Ок і розмістити її праворуч

butOk = new Button ();
butOk.Size = new Size (112,32);
butOk.Dock = DockStyle.Right; // праворуч
butOk.Text = "Ok";

• для порожнього простору між кнопками праворуч додати ще одну панель

}

```
p0 = new Panel (); // ця панель задає відстань між кнопками
p0.Size = new Size (112,32);
p0.Dock = DockStyle.Right;
```

• праворуч додаємо кнопку Cancel

```
butCancel = new Button ();
butCancel.Size = Size (112,32);
butCancel.Dock = DockStyle.Right;
butCancel.Text = "Cancel";
```

• присвоїти кнопкам значення, що повертаються, і зв'язати кнопки з клавішами

```
butOk.DialogResult = DialogResult.OK;
butCancel.DialogResult = DialogResult.Cancel;
DialogResult = DialogResult.Cancel;
AcceptButton = butOk; // натискання клавіші Enter як на OK
CancelButton = butCancel; // натискання клавіші Esc як на Cancel
```

• у методі Main створити об'єкт створеного класу та отримати натиснуту кнопку.

```
OkCancel w = new OkCancel ();
DialogResult rc = w.ShowDialog ();
Console.WriteLine ( "result is {0}", rc);
```

3.3 ЛАБОРАТОРНА РОБОТА: ДІАЛОГОВЕ ВІКНО ВВОДУ ДАНИХ

Зміст заняття: Створення модальних вікон.

Розробити клас для демонстрації питання оператору з декількома полями введення і двома кнопками Так - Ні або Ok - Cancel і додаток для демонстрації вікна цього класу.

- Клас для демонстрації вікна з декількома полями вводу потрібно успадковувати від класу з попередньої лабораторної роботи.

- Використовувати шаблон словника Dictionary <TextBox, type>, де type - певний тип для опис поля введення в діалоговому вікні (повинен містити принаймні назву поля вводу, значення за замовчуванням, поле об’єкта для введеного значення).

- Познайомитися з методами, які приймають змінну кількість фактичних параметрів.

- Познайомитися з подією Click.

- Створити метод-обробник для делегата Click.

Корисна інформація при розробці програми для лабораторної роботи.

Оскільки в попередній лабораторній для горизонтального розміру трьох керуючих елементів використовувалося число 112 (з розрахунком на екрани хорошого розподілення), то для згаданих елементів виберемо число 3*112/2 = 168.

```
}
```

```
t1.Size = new Size (162, 32);
```

Отже, для виконання лабораторної потрібно:

• створити клас для опису полів введення fld. Клас буде містити назву поля введення, видимого оператору (тип Label); рядок, в який потрапить значення, введене оператором в поле введення (тип TextBox); рядок з замовчуваним значенням для поля введення

```
public class fld {
public string name; /// назвf поля
public string value; /// місце для отримання значення
public string def; /// значення за замовчуванням
public fld (string nm, string d = "") {
name = nm;
value = "";
def = d; }
}
```

• у методі Main створити два об'єкти класу fld і вікно класу inWnd. Побудувати вікно для введення цих параметрів і продемонструвати введені значення після натискання на кнопку Ок:

```
fld a = new fld ( "Ім’я", "Вася");
fld b = new fld ( "Прізвище", "Пупкін");
inWnd w = new inWnd ( "Вікно для входу у додаток", a, b);
DialogResult rc = w.ShowDialog ();
if (rc == DialogResult.rc)
  { Console.WriteLine ( "У додаток зайшов користувач: {0} / {1}",
a.value, b.value); }
```

• клас для введення значень inWnd успадковувати від класу ОкCancel (Див. 3.2). Для підтримки декількох полів введення в клас потрібно додати список пар TextBox і fld:

```
using System.Collections.Generic;
public class inWnd: OkCancel {
Dictionary <TextBox, fld> flds = null;
}
```

Він буде використовуватися в обробнику натискання на кнопку для перенесення значень з полів введення в параметри вікна типу fld і відновлення замовчуваних значень.

Для випадку введення двох полів конструктор inWnd повинен мати три параметра:

```
public inWnd (string tit, fld i, fld f) {
 Text = tit; // заголовок вікна
 flds = new Dictionary <TextBox, fld> (); // створити список пар
}

}
```

• в клас inWnd додати метод створення нового поля

```
public TextBox addFld (Int n, fld par)
{ // метод додає в вікно нове поле введення
 return null;
}
```

• Після цього в конструктор додамо два виклики цього методу

```
flds.Add (addFld (0, i), i);  // додали перше поле введення
flds.Add (addFld (1, f), f);  // додали друге поле введення
```

• пишемо тіло методу addFld, в якому для додавання одного поля потрібно два об'єкти PanеI, об'єкт Label - мітка і об'єкт TextBox - поле введення. Метод буде повертати створений TextBox:

Перша панель містить панель (щоб зафіксувати висоту), мітку і поле введення.

```
Panel p = new Panel (); // панель містить мітку і поле введення
p.Size = new Size (1, 32);
p.AutoSize = true;
p.Dock = DockStyle.Top;
p.Padding = new Padding (0,0,0,8); // додати порожній достатньо
// інтервал між мітками
```

У неї додаємо другу панель, притискаючи вгору:

```
Panel p1 = new Panel (); /// панель для висоти
p1.Size = new Size (1, 32);
p1.Dock = DockStyle.Top; /// <--------
p.Controls.Add (p1); /// додати у першу
```

• додаємо мітку з назвою поля вводу, притискаючи її праворуч:

```
l1 = new Label ();
l1.Size = new Size (162, 32) ;;
l1.Text = par.name; /// назва з параметру
l1.Dock = DockStyle.Right;
p.Controls.Add (l1);
```

• додаємо поле вводу, притискаючи праворуч:

```
t1 = new TextBox ();
t1.Size = new Size (162, 32);
t1.TabIndex = n; // задати порядок проходу по полях вводу
// при натисканні на клавішу Tab
t1.Text = par.def; /// привласнили замовчуване значення
// полю вводу
t1.Dock = DockStyle.Right;
p.Controls.Add (t1);
```

}

Перше поле p створено, додаємо його в список керуючих елементів вікна:

```
Controls.Add (p);
return t1; // повернути TextBox
```

• у конструктор inWnd додати обробник натискання на кнопку Ок:

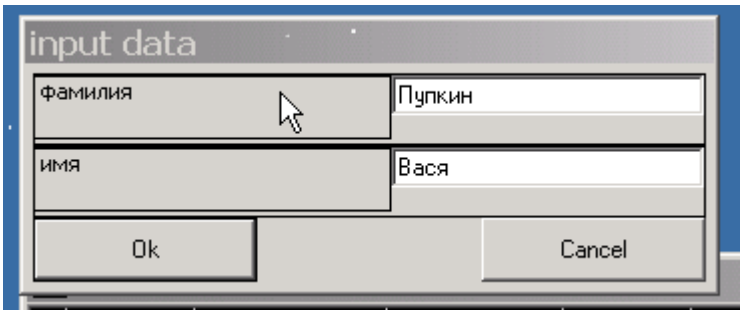
```
public inWnd (string tit, fld i, fld f) {
....
butOk.Click + = _KeyDown;
}
```

• тіло обробника, природно, потрібно додати в клас inWnd:

```
void _KeyDown (object sender, EventArgs e) {
TextBox tb;
fld f;
foreach (KeyValuePair <TextBox, fld> Item in flds)
{
 tb = Item.Key;
 f = Item.Value;
 f.value = tb.Text; // введений текст потрапляє в параметр вікна
 tb.Text = f.def; // відновлюється значення за замовчуванням.
}
}
```

• схожий обробник для відновлення стандартних налаштувань бажано додати або до натискання на кнопку Cancel, або до події активізації вікна Activate.

Якщо, після всіх цих нечуваних зусиль, вийшло вікно на зразок:



Вікно вводу даних

то це означає, що можна створювати діалогові вікна для вводу даних користувачем додатку.

Отриманий клас можна використовувати в усіх лабораторних роботах курсу.

```
}
```

3.4 ЛАБОРАТОРНА РОБОТА: МЕНЮ У ГОЛОВНОМУ ВІКНІ ДОДАТКУ

Зміст заняття: Багатодокументний інтерфейс.

Створити додаток з головним вікном додатка, що містить меню з елементами File, Exit, Work, About і статус - рядок. Додаток має використовувати статус - рядок для відображення свого стану. Нажаніе на Exit має закривати додаток. Натискання на About - призводить до появи діалогового вікна з довідкою про програму.

- Познайомитися з класами MainMenu, MenuItem.
- Познайомитися з класами StatusStrip, ToolStripStatusLabel.
- У класі вікна створити три методи обробника для подій Click в різних елементах меню (у цьому випадку відсутня необхідність у використанні в обробнику параметру obiect sender).

Корисна інформація при розробці програми для лабораторної роботи.

Перед початком виконання лабораторної роботи пригадаємо, що для демонстрації головного вікно додатка необхідно використовувати метод

Application.Run (new Form ());

Головне вікно програми зазвичай містить меню з наступних елементів: Exit – Вихід (піделементи елемента меню File); крайній зліва елемент меню File – зазвичай містить декілька піделементів для виходу з програми, відкриття файлу, закриття файлу і так далі; крайній праворуч елемент меню About – Про себе (містить інформацію про призначення та використання програми). Внизу буде розміщений статус - рядок для відображення стану програми. Зазвичай оператор зможе змінювати розмір головного вікна програми.

У процесі розробки

- спочатку в області видимості класу вікна оголосимо об'єкт статус-рядок, керуючий елемент класу StatusStrip:

```
StatusStrip statStrip;
```

- У конструкторі вікна створюємо керуючий елемент Menu:

```
Menu = new MainMenu ();
```

- Мінімальний набір необхідних елеметов меню:

```
MenuItem FileIt = new MenuItem ( "File");
MenuItem workIt = new MenuItem ( "work");
MenuItem AboutIt = new MenuItem ( "About");
MenuItem ExitIt = new MenuItem ( "Exit");
```

- Додаємо їх (крім Exit) до головного меню:

```
Menu.MenuItems.Add (FileIt);
Menu.MenuItems.Add (workIt);
Menu.MenuItems.Add (AboutIt);
```

- Додаємо Exit до елементу File:

```
FileIt.MenuItems.Add (ExitIt);
```

}

- У конструтора вікна залишилося додати статус-рядок з двох назв
```
statStrip = new StatusStrip ();
ToolStripStatusLabel
statLabel = new ToolStripStatusLabel ();
statStrip.Items.Add (statLabel);
statLabel = new ToolStripStatusLabel ();
statStrip.Items.Add (statLabel); ///
Controls.Add (statStrip);
statStrip.Items [0] .Text ="вікно готове";
statStrip.Items [1] .Text ="дочірніх вікон немає";
```
- Далі, потрібно створити два обробника натискань на елементи меню (Click):

один - для елементу About:
```
void aboutF (object sender, EventArgs a)
 {
  Console.WriteLine ( "about");
  statStrip.Items [1] .Text = "відкрили вікно About";
  MessageBox.Show ( "This programm was made by ImiaRek, Kiev 201 ?.");
  statStrip.Items [1] .Text ="дочірніх вікон немає";
 }
```
другий – для закриття форми (Exit):
```
void exitF (object sender, EventArgs x)
{
Console.WriteLine ( "FormClosing");
Close ();
}
```
- У конструкторі додамо їх до подій – змінних:
```
AboutIt.Click += aboutF;
ExitIt .Click += exitF;
```

3.5 ЛАБОРАТОРНА РОБОТА: ПАНЕЛЬ ІНСТРУМЕНТІВ ДОЧІРНЬОГО ВІКНА З ПРЕДСТАВЛЕННЯМ ТАБЛИЧНИХ ДАНИХ

Зміст заняття: Подання табличних даних.

До додатку з лабораторної 3.4 додати клас дочірнього вікна для демонстрації вмісту CSV файлу за допомогою керуючого елемента DataGridView. Головне вікно додатку із попередньої лабораторної залишається, а по натисканню на елемент меню Work головне вікно, дочірнє вікно повинне містити поки не використану панель інструментів з набором кнопок: Відкрити, Перечитати, Редагувати, Додати, Видалити, Сортувати, Фільтрувати, Експортувати, Закрити.

- Познайомитися з класами ToolBar, ToolBarButton (або ToolStrip).
- Створити метод обробник для події Click на панелі інструментів, потрібно обробляти натискання на кнопку Відкрити, а потім кнопку Закрити.

```
}
```

- Для відкриття файлу використовувати стандартне діалогове вікно OpenFileDialog, пошук файлів починати з каталогу з якого було запущено програму (а не з диска C:).

- Створити метод обробник для події Click на елемент меню з головного вікна програми. Відкриваюче дочірнє вікно повинно змінити стан статус рядка головного вікна програми.

- Познайомитися з класом DataGridView і навчитися заповнювати його даними з будь-якого CSV файлу.

Корисна інформація при розробці програми для лабораторної роботи.

Лабораторна досить об'ємна і, тому, пояснення розділяється на дві частини. У першій частині буде створений і заповнений даними керуючий елемент для подання таблиць, в другій, до цього вікна додається керуючий елемент панель інструментів для майбутнього редагування таблиці.

I. Вікно з поданням табличних даних

- Підключаємо до проекту бібліотеку, що динамічно завантажується, System.Windows.Forms.dll. При компіляції проекту з командного рядка, команду csc /out:a.exe *.cs
  замінити на csc /out:a.exe /r:System.Windows.Forms.dll *.cs

- Іменовану область видимості System.Windows.Forms додаємо до перегляду за замовчуванням. У код додати рядок

```
using System.Windows.Forms;
```

- Перед методом Main додати атрибут статичної однонитевої програми:

```
[STAThread]
static void Main (string [] args)
```

- Клас OpenFileDialog використовуємо для вибору потрібного файлу з даними приблизно у такому вигляді:

```
OpenFileDialog o1 = New OpenFileDialog (); // створити змінну
o1.Filter = "Txt files (* .txt) | * .txt | All files (*. *) | *. *";
                                        // відфільтрувати файли директорії
o1.InitialDirectory = Path.GetDirectoryName
                (Assembly.GetExecutingAssembly (). Location );
 //початковий каталог для перегляду повинен збігатися з місцем,
 // звідки виконується збірка
if (o1.ShowDialog () == DialogResult.OK)
    {// якщо файл був заданий, отримати його ім'я
      string fnm = openFileDialog1.FileName;
      // продовжувати роботу тут }
```

- Створюємо клас для демонстрації майбутнього вікна з керуючим елементом DataGridView:

```
class wnd: Form {
 string flNm;
 protected DataGridView dgv; //це поле буде доступно нащадкам класу
}
```

```
        public wnd (string fileName) {
         flNm = fileName;
         dgv = new DataGridView ();
         dgv.ReadOnly = true; // заборонити редагування осередків оператором
         dgv.AllowUserToAddRows=false;//заборонити додавання рядків
          dgv.Dock = DockStyle.Fill; // заповнити вікно
         dgv.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
         Controls.Add (dgv);
        }
    }
```

- Створюємо обробник для заповнення DataGridView:

```
   void fLoad (object sender, EventArgs a) {
    Console.WriteLine ( "Load");
    ReadFile (flNm);
    }
```

- Створюємо метод для читання файлу:

```
    public void ReadFile (string path) {
    string str;
    string [] fields;
    bool firsttime = true;
    using (StreamReader sr = new StreamReader (path)) {
    while ((str = sr.ReadLine ())! = null) {
            fields = str.Split ( ';');// крапка з комою - роздільник записів
            if (fields.Length <= 0) // пропустимо порожні рядки
    continue;
    if (firsttime) {//на першому рядку дізнатися кількість колонок
    dgv.ColumnCount = fields.Length;
    for (int i = 0; i <fields.Length; i ++) {
    dgv.Columns [i] .Name = String.Format ( "Pole {0}", i + 1);
    }
    firsttime = false;
    }
            dgv.Rows.Add (fields);// додати новий рядок
    }
  }
}
```

- У констуктор wnd до події Load додати створений обробник:

```
Load + = fLoad;
```

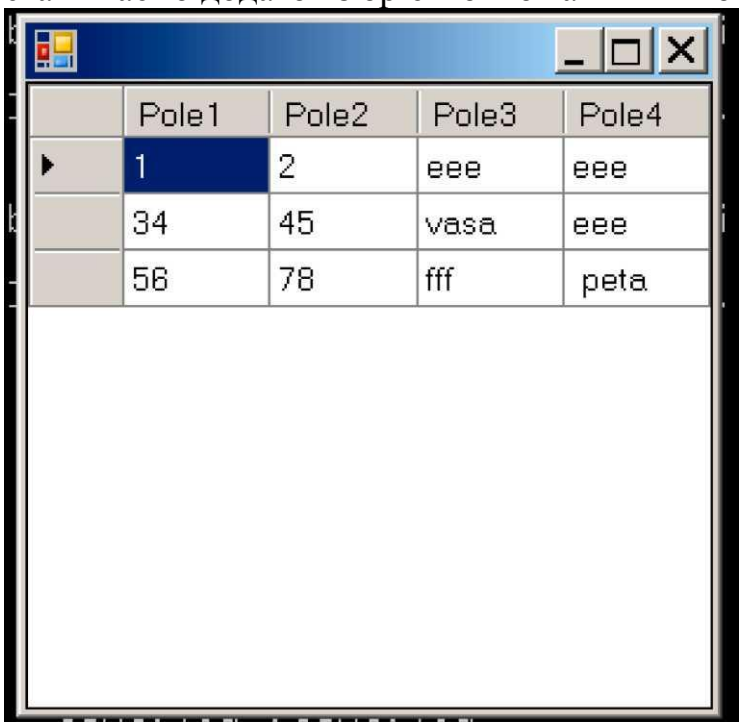
Подія Load виникає при появі перший раз вікна на екрані комп'ютера при виконанні програми. Для кожної появи будь-якого вікна при кожному виконанні додатку існує одна подія Load.

```
}
```

•У методі Main створюємо вікно розробленого класу wnd і демонструємо його в якості головного вікна програми:

```
{// якщо файл не був заданий, отримати його ім'я
string fnm = openFileDialog1.FileName;
// продовжувати роботу тут
Application.Run (new wnd (fnm));
}
```

На цьому етапі маємо додаток з орієнтовно таким вікном виводу:



Мал.3.5.1 Вікно з поданням табличних даних

II. Панель інструментів дочірнього вікна

У попередній частині роз'яснювалися початкові дії по наповненню керуючого елемента DataGridView. Далі необхідно додати до вікна два керуючих елемента: вже раніше використаний в лабораторній 3.4 статус-рядок - StatusStrip і новий елемент - смуга інструментів ToolBar.

Для розробки прикладу з поданням табличних даних необхідно:

• З класу wnd успадковувати клас wnd2:

```
public class wnd2: wnd
{
ToolBar tb = new ToolBar ();
StatusStrip statStrip = new StatusStrip () ;;
public wnd2 (string fnm): base (fnm)
        {}
}
```

• Замінити створення вікна в методі Main:

```
// попереднє вікно класу батька без керуючих елементів
// Form y = new wnd (fnm);
}
```

```
// розробляється вікно класу спадкоємця з керуючими елементами
  Form y = new wnd2 (fnm);
```

• У конструктор wnd2 додати створення та ініціалізацію статус-рядку:

```
ToolStripStatusLabel
statLabel = new ToolStripStatusLabel ();
statStrip.Items.Add (statLabel);
Controls.Add (statStrip);
statStrip.Items [0] .Text = "datagrid window is ready";
```

• У конструктор wnd2 додати розміри для майбутніх кнопок керуючого елемента ToolBar (смуга інструментів), що б зробити їх однаковими

```
tb.ButtonSize = new System.Drawing.Size ((int) (200/3), (int) (40));
```

• У конструктор wnd2 створити кілька кнопок для керуючого елемента ToolBar (це об'єкти класу ToolBarButton). Задати у них текст підказки (поле класу ToolTipText):

```
ToolBarButton tlbExit = new ToolBarButton ( "Exit");
tlbExit.ToolTipText ="закрити вікно таблиці";
ToolBarButton tIns = new ToolBarButton ( "Insert");
tIns.ToolTipText ="додати запис";
ToolBarButton tEdit = new ToolBarButton ( "Edit");
tEdit.ToolTipText ="редагувати запис";
ToolBarButton tDel = new ToolBarButton ( "Delete");
tDel.ToolTipText ="видалити запис";
ToolBarButton tSave = new ToolBarButton ( "Save");
tSave.ToolTipText = "зберегти зміни";
ToolBarButton tExp = new ToolBarButton ( "Export");
tExp.ToolTipText ="експортувати таблицю";
```

• Додати їх до смуги інструментів

```
tb.Buttons.AddRange (new ToolBarButton [] {
    tIns , tEdit , tDel , tSave , tExp , tlbExit }
);
```

• Прикріпити смугу інструментів до верхнього краю вікна

```
tb.Dock = DockStyle.Top;
Controls.Add (tb);
```

• Створити ще один обробник події Load (перший буде викликатися з класу wnd), який запише в статус-рядок кількість рядків керуючого елемента DataGridView:

```
void fLoad (object sender, EventArgs a)
{
statStrip.Items [0] .Text = String.Format ( "{0} records has been red",
dgv.Rows.Count);   }
```

• Додати обробник до делегату - полю події Load

```
Load+=fLoad;
```

```
}
```

• Створити обробник події Click на смузі інструментів

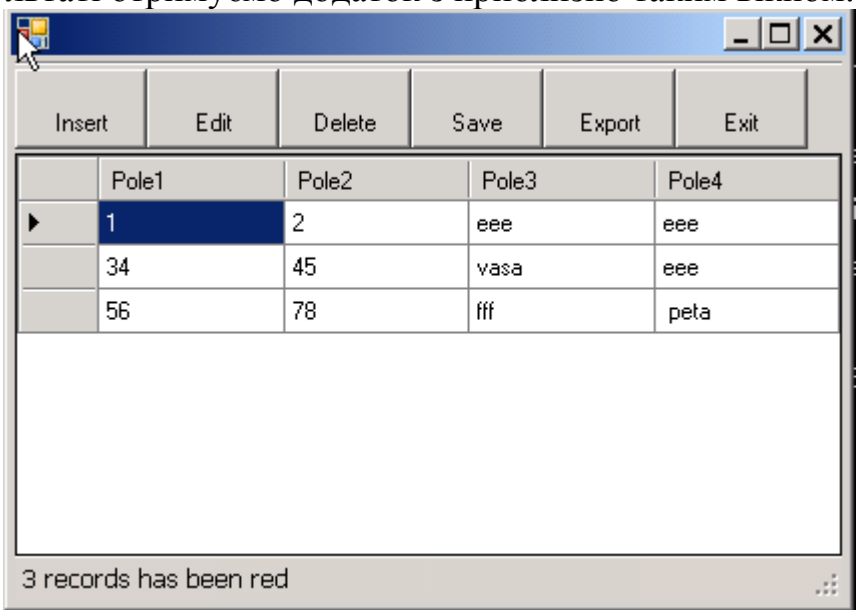
```
void ToolBarButtonClick (object sender, ToolBarButtonClickEventArgs e)
{
string bNm = e.Button.Text;
statStrip.Items [0] .Text = String.Format ( "You've pressed {0} button", bNm);
if (bNm == "Exit")
 Close ();
else if (bNm == "Open") ;
else {
 MessageBox.Show ( "action not ready!", "Warning"); }
}
```

За кодом прикладу зрозуміло, що клас параметрів даного обробника ToolBarButtonClickEventArgs дозволяє дізнатися назву відповідної кнопки і викликати код, призначений для неї.

• Додати обробник до делегату - полю смуги інструментів tb події ButtonClick

```
tb.ButtonClick += new ToolBarButtonClickEventHandler (ToolBarButtonClick);
```

В результаті отримуємо додаток з приблизно таким вікном:



Мал.3.5.2 Вікно для демонстрації табличних даних

3.6 ЛАБОРАТОРНА РОБОТА: ВІКНО РЕДАГУВАННЯ ТАБЛИЦЬ

Зміст заняття: Перегляд і редагування табличних даних.

Створити додаток з головним вікном додатка 3.5 і розробити дочірнє вікно для перегляду таблиці, редагування записів з таблиці за допомогою діалогового вікна додатку лабораторної 3.3. Дані для таблиці повинні знаходитися в CSV-файлі з однім типом кодування (866, 1251, UTF8, UTF16).

- Продовжити засвоєння роботи з класом DataGridView.
- Кодування задавати двома способами: по старому: ключем із

```
}
```

командної строки, по новому: при помощі діалоговога вікна з головного меню, в яке добавити елемент Parameters (між Work і About).

- Рядки з таблиці(записи) редагувати за допомогою діалогового вікна.
- Стандартні вікна для відкриття і збереження файлів повинні за замовчуванням показувати каталог із якого був запущений додаток.
- Використовувати класи для кодування символів національних алфавітів (кирилиці) (Encoding).
- Переглянути особливості використання колекцій в .Net.
- Створити методи обробники для читання, сортування, редагування, експортування та збереження таблиці в файлі.
- При натисканні на елемент меню Work головного вікна бажано відкривати не більше одного дочірнього вікна для перегляду вмісту таблиці.

Корисна інформація при розробці програми для лабораторної роботи.

3.6.1. Редагування таблиці, частина 1

Розробка програми для редагування таблиці починається з додавання фукнціоналу для кнопки Insert.

- Скопіювати в окремий каталог файли з п.3.5;
- В цей же каталог з п.3.3 скопіювати файли inWnd.cs і OkCancel.cs;
- У класі wnd2 до полів tb і statStrip та до методу ToolBarButtonClick дописати ключове слово protect;
- Створити файл wnd3.cs:

```
namespace dtGrd
{
 public class wnd3: wnd2
 {
    public wnd3 (string fnm): base (fnm) {
    // видалити обробник батьківського класу
    tb.ButtonClick - = base.ToolBarButtonClick;
    }
  }
}
```

- Підключити до нього область видимості з лабораторної «Вікно вводу даних»:

```
using laba3;
```

- У методі Main виконати вікно нового класу:

```
Application.Run (new wnd3 (fnm));
```

- Кнопки керуючих елементів на панелі інструментів перестають реагувати на натискання. До класу wnd3 додати його обробник

```
void ToolBarButtonClick3 (object sender, ToolBarButtonClickEventArgs e)
 {
 string bNm = e.Button.Text;
 statStrip.Items [0] .Text= String.Format ( "You’ve pressed {0} button", bNm);

}
```

```
        if (bNm == "Exit")
        Close ();
        else if (bNm == "XXX")
        ;
        else {
          // викликати батьківський обробник
          base.ToolBarButtonClick(Sender, e);
    }
  }
```

в якому викликається обробник батьківського класу wnd2 і додати його до події

```
        tb.ButtonClick + = ToolBarButtonClick3;
```

Це відновлює реакцію додатка при натисканні кнопок панелі інструментів. Для забезпечення можливості редагування таблиці з будь-якою кількістю колонок в клас inWnd додати конструктор для передачі в вікно масиву полів типу fld:

```
        public inWnd (string Title, fld [] ps): base (Title)
        {
         flds = new Dictionary <TextBox, fld> ();
         // створити список пар вигляду: поле введення + параметр
         for (int i = 0; i <ps.Length; i ++)
         //в список пар додати поле вводу, зроблене для
         flds.Add (addFld (i, ps [i]), ps [i]);
         // чергового параметра + сам параметр
         butOk.Click + = _KeyDown;
        }
```

- В клас wnd3 додати метод для підготовки:

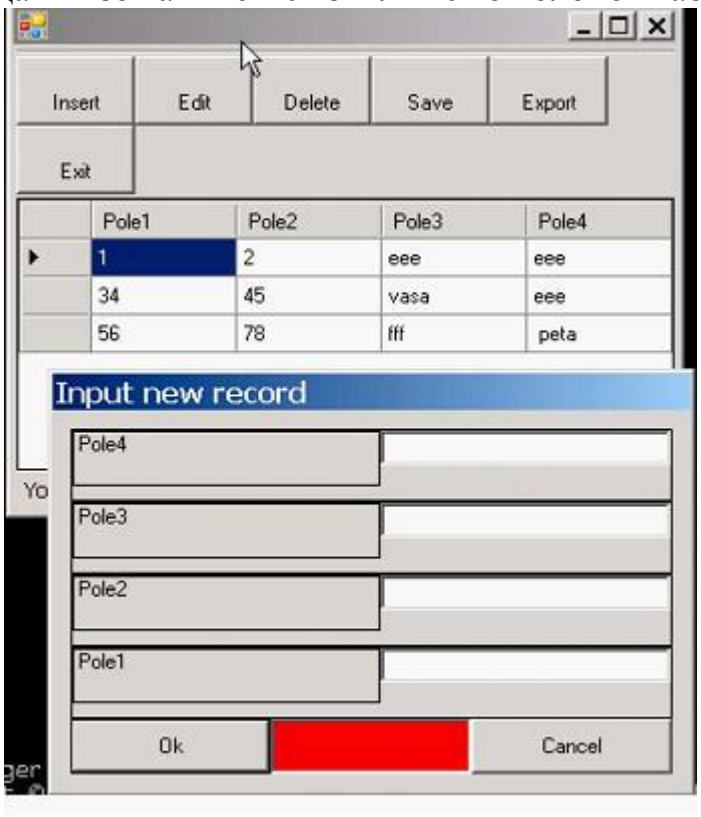
```
        public void doIns () {
         fld f;
         List <fld> flds = new List <fld> ();
         for (int i = 0; i <dgv.ColumnCount; i ++) {
         f = new fld (dgv.Columns [i] .Name, "");
         flds.Add (f);
        }
         if (flds.Count> 0) {
         Form w = new inWnd ( "Input new record", flds.ToArray ());
         DialogResult rc = w.ShowDialog ();
        }
        else {
         statStrip.Items [1] .Text= String.Format ( "nothing to do!");
        }
   }
```

- Для виклику вікна вводу даних при натисканні на кнопку "Insert" потрібно змінити обробник ToolBarButtonClick3:

```
  }
```

```
string bNm = e.Button.Text;
if (bNm == "Exit")  Close ();
else if (bNm == "Insert") doIns ();
else {// викликати батьківський обробник
 base.ToolBarButtonClick (Sender, e);
}
```

• Після натискання на кнопку Insert на панелі інструментів вікна перегляду таблиці має з'явитися вікно вводу даних. Причому кількість полів у вікні введення даних збігатиметься з кількістю колонок таблиці.



Додавання запису

Першу частину створення додатку закінчено.

3.6.2 Редагування таблиці, частина 2

У другій частині лабораторної забезпечимо, щоб при натисканні на кнопку Ок дані з вікна вводу потрапляли в таблицю, а після натискання на кнопку Save смуги інструментів - в вихідний файл. Крім того, внесемо незначні зміни у вигляд таблиці (для вивчення нових властивостей класу DataGridView) і додамо реакцію додатка при натисканні на клавіші клавіатури.

Продовжуємо розробку програми редагування CSV-файлу.

• Маємо прибрати з таблиці найлівішу сіру колонку без назви з чорним трикутником і сине виділення комірки та замінити на широке сине виділення запису:

```
dgv.SelectionMode
 = DataGridViewSelectionMode.FullRowSelect; // виділяємо весь запис

}
```

```
dgv.RowHeadersVisible = false; // зліва видаляємо зайву колонку
```

• Доопрацьовуємо метод doIns з класу wnd3 так, щоб при натисканні на кнопку Ок вікна вводу даних, введений рядок потрапив в таблицю:

```
DialogResult rc = w.ShowDialog ();
 if (rc == DialogResult.OK) {// якщо натиснули Ок
   statStrip.Items [0] .Text = String.Format ( "insert after {0} rec",
     Dgv.CurrentRow.Index);
   string [] fs = new string [flds.Count];// створили масив для вставки
                                          // нового запису в таблицю
   for (int j = 0; j <flds.Count; j ++) {// заповнили масив новими полями
     fs [j] = flds [j] .value; }
   dgv.Rows.Insert (dgv.CurrentRow.Index, fs); // вставили нові поля
   }                                                  //в таблицю
```

Рядок додано в таблицю.

• У найпершому батьківському класі wnd для демонстрації таблиці з лабораторної 3.5 методам з класів спадкоємців (wnd3) відкрити доступ до поля з ім'ям файлу продемонстрованої таблиці:

```
protected          // це поле повинно бути доступно
string fileName; // для обробника кнопки збереження
                   // у вікні редагування таблиці
```

• Створити метод doSave для збереження таблиці в файл, з якого вона була прочитана:

```
public void doSave () {
  string ss = csvExport (fileName, // поле класу батьківського класу wnd
  ’;’, Dgv);
 Console.WriteLine ( "Save: ’{0}’", ss);
 statStrip.Items [0] .Text = ss;
}
```

Передбачається, що текст методу csvExport, який переносить дані з керуючого елемента dgv в csv-файл з іменем fileName і роздільником ';' , теж потрібно написати.

• Додати в обробник ToolBarButtonClick3 виклик доданого методу doSave (метод doIns вже викликається в першій частині):

```
else if (bNm == "Save") doSave ();
```

Можна перевірити роботу кнопок панелі інструментів Insert і Save.

• Додати в обробник ToolBarButtonClick3 виклик методу doEdit

```
else if (bNm == "Edit") doEdit ();
```

а в клас заглушку методу:

```
public void doEdit () {}
```

• Для редагування спочатку перевірити в doEdit чи є записи в таблиці:

```
public void doEdit () {

}
```

```
string ss = "";
   if (dgv.RowCount> 0) {
     // редагування тут
    } else ss = "nothing to dp!";
  statStrip.Items [0] .Text = ss;
}
```

• Підготувати список полів для створення вікна редагування запису:

```
if (dgv.RowCount> 0) {
 DataGridViewRow dgvR = dgv.Rows [dgv.CurrentRow.Index];
 //з поточного рядка беремо
 DataGridViewCellCollection row = dgvR.Cells;
 //значення за замовчуванням
 List <fld> flds = new List <fld> ();
 fld f;
 for (int i = 0; i <dgv.ColumnCount; i ++) {
  f = new fld (dgv.Columns [i] .Name
       , (Row [i]). Value.ToString ());// значення за замовчуванням
       flds.Add (f); // використовуються для створення поля
   }
  if (flds.Count> 0) {
        // редагування тут
   }
  else
    ss = "Error: there are not any column!";
```

• Показати вікно і отримати з нього дані, введені оператором:

```
 if (flds.Count> 0) {
  Form w = new inWnd ( "Edit selected record", flds.ToArray ());
  DialogResult rc = w.ShowDialog ();
  if (rc == DialogResult.OK) {// якщо ок
  for (int kk = 0; kk <flds.Count; kk ++) {// перенести значення в таблицю
   dgvR.Cells [kk] .Value = flds [kk] .value;
   }
  }
 }
```

Рядки вже будуть редагуватися і, отже, їх можна зберігати в файлі.

• Доопрацювати натискання на DoubleClick на запису. Це аналог натискання на кнопку Edit. Делегат - обробник:

```
protected
 void dc2 (object sender, DataGridViewCellEventArgs e)
 {
  Console.WriteLine ( "CellDoubleClick: selrow / row: {0} / {1}"
  , Dgv.CurrentRow.Index, e.RowIndex);
  if (e.RowIndex> = 0) {

}
```

```
doEdit ();
dgv.CurrentCell = dgv.Rows [e.RowIndex] .Cells [0];
}
}
```

В конструкторі додати обробник для події DoubleCIick в комірці таблиці:

```
dgv.CellDoubleClick + = dc2;// редагування запису по Enter
```

• Обробник для натискання на клавішу Enter:

```
void _PreviewKeyDown (object sender, PreviewKeyDownEventArgs e)
{
if (e.KeyCode == Keys.Enter)
{
Console.WriteLine ( "CellDoubleClick: selrow / row: {0}"
, Dgv.CurrentRow.Index);
doEdit ();
}
}
```

• Додати обробник події KeyDown:

```
dgv.PreviewKeyDown + = _PreviewKeyDown;
```

• Натискання на клавішу Enter стандартно призводить до переходу на наступний рядок. Якщо ця поведінка не подобається, можна її позбутися:

```
void _KeyDown (object sender, KeyEventArgs e) {
if (e.KeyCode == Keys.Enter) {
e.SuppressKeyPress = true;
}
}
i
dgv.KeyDown + = _KeyDown;
```

Таблиця повинна редагуватися і зберігатися. Обробку натискання на клавішу Insert (вставка запису в таблицю) зробити самостійно.

3.7 ЛАБОРАТОРНА РОБОТА: БАГАТОДОКУМЕНТНИЙ ІНТЕРФЕЙС

Зміст заняття: Багатодокументний інтерфейс.

Створити додаток для перегляду і редагування бази даних Постачальників див. [8, с.~257].

- У додатку використовувати всі раніше вивчені класи.
- У головному меню додати елемент меню Windows, в який повинні автоматично додаватися заголовки всіх відкритих вікон.

Корисна інформація при розробці програми для лабораторної роботи.

```
}
```

При розробки програми для лабораторної має бути використаний код додатків з попередніх лабораторних робіт - 3.4 (Меню в головному вікні програми) і 3.6 (Редагування табличних даних) для побудови додатку з мультидокументним інтерфейсом. Додаток повинен показувати таблиці бази даних постачальників наступного змісту:

самі постачальники:

```
---- File:./date/date/s.csv
    S1,Smith,20,London
    S2,Jones,10,Paris
    S3,Black,30,Paris
    S4,Clark,20,London
    S5,Adams,30,Athens
---- End Of File:./date/date/s.csv
```

деталі, які вони поставляли:

```
---- File:./date/date/p.csv
    P1,Nut,Red,12.0,London
    P2,Bolt,Green,17.0,Paris
    P3,Screw,Blue,17.0,Rome
    P4,Screw,Red,14.0,London
    P5,Cam,Blue,12.0,Paris
    P6,Cog,Red,19.0,London
---- End Of File:./date/date/p.csv
```

проекти, для яких виконувалися поставки деталей:

```
---- File:./date/date/j.csv
    J1,Sorter,Paris
    J2,Display,Rome
    J3,OCR,Athens
    J4,Console,Athens
    J5,RAID,London
    J6,EDS,Oslo
    J7,Tape,London
---- End Of File:./date/date/j.csv
```

самі поставки:

```
---- File:./date/date/spj.csv
    S1,P1,J1,200
    S1,P1,J4,700
    S2,P3,J1,400
    S2,P3,J2,200
    S2,P3,J3,200
    S2,P3,J4,500
    S2,P3,J5,600
    S2,P3,J6,400
    S2,P3,J7,800
}
```

```
S2,P5,J2,100
S3,P3,J1,200
S3,P4,J2,500
S4,P6,J3,300
S4,P6,J7,300
S5,P2,J2,200
S5,P2,J4,100
S5,P5,J5,500
S5,P5,J7,100
S5,P6,J2,200
S5,P1,J4,100
S5,P3,J4,200
S5,P4,J4,800
S5,P5,J4,400
S5,P6,J4,500
---- End Of File:./date/date/spj.csv
```

Перевіримо, що у якості роздільника в програмі і в файлах використовується один і той же символ, і починаємо створювати проект.

- У новий каталог перекопіюємо файли з 3.2.5 - Program.cs (його буде перероблено) і з 3.2.6 - файли wnd.cs і wnd2.cs.
- В файл головного вікна програми Program.cs додати підключення іменованої області видимості двох останніх файлів:

```
using dtGrd;
```

- У класі Program завести статичну змінну типу Form і у ній створити головне вікно програми:

```
static public Form mWin;
[STAThread]
static void Main (string [] args) {
 mWin = new x ();
 mWin.IsMdiContainer = true;
 Application.Run (mWin);
}
```

(тут було встановлено ознаку контейнера мультидокументного інтерфейсу).

- Змінити меню головного вікна програми на таке:

```
File Date Windows About
Exit Supplier
Part
Project
Delivery
```

При цьому:

```
MenuItem SupplierIt = new MenuItem ( "Supplier");
MenuItem PartIt = new MenuItem ( "Part");
```

```
}
```

```
MenuItem ProjectIt = new MenuItem ( "Project");
MenuItem DeliveryIt = new MenuItem ( "Delivery");
MenuItem WinIt = new MenuItem ( "Windows");
```

(подробиці створення головного меню програми див. 3.2.)

•Додамо обробники події Click нових елементів меню:

```
SupplierIt.Click + = SupplierF;
PartIt .Click + = PartF;
ProjectIt .Click + = ProjectF;
DeliveryIt.Click + = DeliveryF;
```

•Додамо чотири практично однакових обробникі:

```
void SupplierF (object sender, EventArgs a)  {
  Console.WriteLine ( " Постачальники ");
  statStrip.Items [1] .Text = " вікно постачальників не готове ";
 }
```

• У кожному з чотирьох обробників задаємо властивість Mdi-Parent:

```
z.MdiParent = Program.mWin;
```

і доопрацьовуємо проект, приблизно, у такому вигляді:

```
Form z = new wnd2 (// вибрати папку, з якої запущено додаток
Path.GetDirectoryName (Assembly.GetExecutingAssembly().Location)
+ "/ P.csv");                    // додати файл поставок
z.Text = " Деталі ";  z.Name = "part";
statStrip.Items [1] .Text = " Деталі завантажені ";
z.Show ();
```

Елементу меню WinIt задаємо ознаку списку дочірніх вікон мультидокументної програми:

```
WinIt.MdiList = true;
```

• Приберемо необхідність багаторазового відкриття списку. В результаті додаток буде мати не більше чотирьох відкритих вікон: одне - для постачальників, одне - деталей і т.д. Для цього в клас головного вікна програми додаємо метод

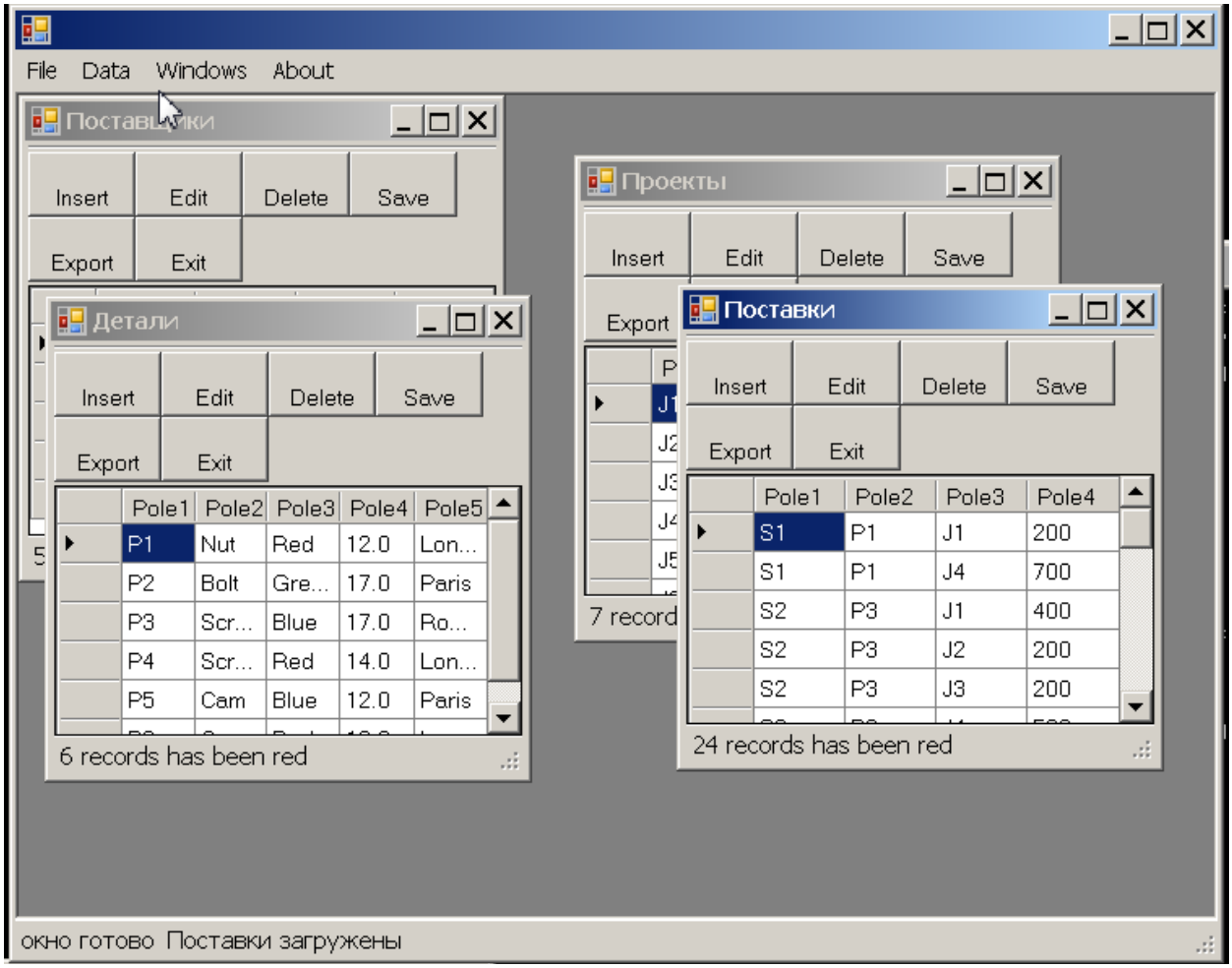
```
bool shouldIOpen (string text){
 for (int i = 0; i < MdiChildren.Count(); i++)  {
  if (this.MdiChildren[i].Text == text)
  { MdiChildren[i].Activate(); return false; }
 }
 return true;
}
```

За допомогою цього методу можна змінити обробників елементів головного меню програми таким чином, щоб вони не відкривали нові вікна, а активізовували вже відкриті. Для прикладу продемонструємо це на обробнику вікна проектів:

```
}
```

```
void ProjectF (object sender, EventArgs a) {
 Console.WriteLine ( " Проекти ");
 string id = "project";
 if (shouldIOpen (id)) {
 statStrip.Items [1] .Text = " вікно проектів не готове ";
 Form z = new wnd2 (
 Path.GetDirectoryName (Assembly.GetExecutingAssembly (). Location)
 + "/ J.csv");
 z.MdiParent = Program.mWin;
 z.Text = " Проекти"";
 z.Name = id;
 z.Show ();
 }
 statStrip.Items [1] .Text = " Проекти завантажені ";
}
```

В результаті створення додатку маємо отримати щось на зразок:



Багатодокументний інтерфейс

Причому, вміст елемента Windows з головного меню буде змінюватися в залежності від кількості відкритих вікон. Побудована програма починає нагадувати додаток для ведення бази даних.

```
}
```

3.8 ЗАВДАННЯ ДЛЯ МОДУЛЬНОЇ РОБОТИ

В якості модульної роботи розробити проект:

1. Віконну версію додатка Каса (видача решти). На відміну від консольної версії програма має конвертувати тестовий файл з описом стану каси в двійковий і навпаки. Для зберігання стану каси використовувати двійковий файл. Мати можливість переглядати поточний стан каси. Вікно додатка з'являється тільки в разі відсутності в командному рядку ключів -txt2bin, -bin2txt і -з MMM.

Запуск додатку для видачі решти нав виконання має вигляд

a.exe [-?] [-txt2bin | -bin2txt] -f state.bin [-з MMM]

-? видача підказки
-txt2bin стан каси зі стандартного вводу конвертувати в файл state.bin
-bin2txt стан каси з state.bin вивести в стандартний потік виводу
-з MMM видати решту в консольному режимі, не показувати вікно додатка
-f state.bin файл з описом стану каси.

2. Віконну версію програми транслітерації імен файлів з заміною спецсимволов (прохід по дереву каталогів).

3. Віконну версію додатка для включення-відключення автозапуску для змінних накопичувачів (USB-флеш-накопичувачів, оптичних дисків (CD-ROM, DVD-ROM)).

4. Додаток для перегляду та редагування бази даних Постачальників (Без журналювання роботи програми), DATE:more.

5. Додаток для руху кнопки Ok, що тікає від покажчика мишки. Додаток має демонструвати таблицю координат кнопки.

6. Віконну версію програми для зміни кодування текстового файлу.

7. Багатодокументний додаток, що повторює інтерфейс системи обмеження доступу з [16, с.~83].

8. Утиліт для виявлення на кадрі смужок однакового кольору (підготовка до виявлення знаків дорожнього руху).

Бажано, що б додатки задовольняли наступним умовам:

• використовувати керуючі елементи Menu + MenuItem (або MenuStrip + ToolStripMenuItem) - для надання команд оператора в додатку.

• використовувати керуючі елементи ToolBar + ToolBarButton (або ToolStrip + ToolStripButton) - альтернативний спосіб надавати команди оператора.

• використовувати керуючий елемент StatusBar (або StatusStrip) - для відображення інформації про стан програми та дочірніх вікон.

• використовувати керуючий елемент DataGridView для відображення табличних даних.

• використовувати мультідокументний інтрефейс для відкриття декількох дочірніх вікон.

}

МОДУЛЬ 4: ДОДАТКОВІ МОЖЛИВОСТІ ПРОГРАМУВАННЯ В .Net

4.1 ЛАБОРАТОРНА РОБОТА: РЕСУРСИ В ДОДАТКАХ

Зміст заняття: Ресурси в додатках.

Створити додаток з мінливим малюнком в головному вікні програми.

- Познайомитися з поняттям ресурси програми.
- Познайомитися з класом ResXResourceWriter, створити файл ресурсів resx.
- Познайомитися з компілятором ресурсів resgen.exe.
- Познайомитися з класами ResourceManager.
- Познайомитися з поняттям таймера (Timer).

Для виконання роботи 4.1 потрібно створити файл ресурсів, скомпілювати його, додати до процесу побудови програми і продемонструвати роботу останнього. Під ресурсом в даному контексті розуміється іконка, покажчик миші, картинка або тестовий рядок. Їх можна не включати в збірку, але, якщо вони потрібні для роботи програми, необхідно їх скопіювати разом з додатком. Доступ до таких ресурсів з додатку буде звичайним читанням файлів. У іншому випадку їх можна включити в збірку: тоді копіювати потрібно лише саму збірку, але доступ прийдеться організувати не через файли, а через менеджер ресурсів.

4.1.1 Побудова додатку з ресурсами

Для виконання лабораторної роботи треба побудувати файл ресурсів (picslst.resx). Це можна зробити декількома способами. Наприклад, у додатку на C# файли ресурсів будь-яких типів створюються об’єктами класу ResXResourceWriter. Об’єкт створюється, до нього додаються ресурси, наприклад, змінні типу Image (використовувані для зберігання картинок), а об’єкт зберігається на жорсткому диску в момент закриття. Файл ресурсів (1pic.resx) з однієї картинкою (some.jpg) можна створити так:

```
ResXResourceWriter rw = new ResXResourceWriter ( "picslst.resx");
rw.AddResource ( "pic1", // ім’я ресурсу !!!!
                         Image.FromFile ( "some.jpg", true));
rw.Close ();
```

Для створення файлу з текстовими ресурсами можна скористатися компілятором ресурсів resgen.exe (див., наприклад, https://docs.microsoft.com/en-us/dotnet/ framework/tools/resgen-exe-resource-file-generator). Припустимо, що файл 1.txt з описом текстового ресурсу ares має вигляд:

```
ares = SomeText_SomeText
```

Текст, наведений у кирилиці, за замовчуванням повинен бути записаний в кодуванні UTF-8. Тоді створення файлу 1.resx буде виконуватися наступним чином:

```
resgen 1.txt 1.resx
```

У вихідному файлі повинна бути присутня секція такого вигляду:

```
}
```

```
<Data name = "ares" xml: space = "preserve">
<value> SomeText_SomeText </ value>
</ Data>
```

Для створення ресурсів зі списку картинок можна також використовувати утиліту [29]

Остаточна компіляція ресурсів для включення їх в збірку (тобто, створення файлу з розширенням .resources) виконується за допомогою команди:

resgen.exe /usesourcepath /compile picslst.resx.

Якщо компіляція ресурсів пройшла без помилок в каталозі повинен з'явитися файл picslst.resources. Його треба підключити до збірки командою:

csc/resource:picslst.resources/out:a.exe *.cs

Побудований надалі додаток a.exe буде містити вихідні картинки в розділі ресурси.

4.1.2 Малювання та доступ до ресурсів додатку

Доступ до ресурсів можна отримати наступним чином

```
using System.Reflection;
class frmMain: Form {
...
System.Resources.ResourceManager rm = New global ::
System.Resources.ResourceManager ( "Picslst",
// відкомпільований файл ресурсів, підключений до збірки
Typeof (frmMain) .Assembly); // клас з методом Main
Image im = (Image) rm.GetObject ( "pic1");
                    // взяти байти за іменем ресурсу !!!!
}
```

Для малювання картинки у вікні, створити метод-обробник для перемальовування вікна. Його ім'я має бути OnPaint і параметр мати тип PaintEventArgs:

```
 protected override void OnPaint (PaintEventArgs ea)
{ Graphics grfx = ea.Graphics;
grfx.DrawImage (im, 10, 10, im.Width, im.Height);
}
```

Об'єкт grfx (Graphics) - це віртуальна поверхня, на якій методом DrawImage малюється раніше прочитана картинка im.

4.2 ЛАБОРАТОРНА РОБОТА: ПРИМІТИВИ ГРАФІКИ

Зміст заняття: Основи програмування графіки.

Створити додаток з немасштабованною кривою в головному вікні програми і намальованим вертикальним і горизонтальним текстом в заданому місці, заданого кольору.

- Познайомитися з подією Paint, класом PaintEventArgs, методом

```
}
```

Invalidate.

- Познайомитися з класом Graphics, з примітивами для малювання ламаних, еліпсів і прямокутників.
- Познайомитися з типами SolidBrush, Pen, Color.
- При заданому ключі -v - підписати номера точок ламаної.

Корисна інформація при розробці програми для лабораторної роботи.

Для виконання лабораторної 4.2 координати пікселів у вікні будуть задаватися цілими числами. При цьому, передбачається, що читання послідовності пар цілих чисел не є проблемою, тому для створення методу

public Point [] // читання масиву точок getPoints (TextReader tr) {}

якій читає зі стандартного вводу масив точок, не потребує пояснень.

Створити чотири файли: Program.cs, sLine.cs, p4sLine.cs і w4line.cs. Область видимості за замовчуванням - line.

В кожному файлі іменовану область видимості System.Drawing додати до перегляду за замовчуванням. Вона містить структуру Point, що представляє собою впорядковану пару цілих чисел - координат X і Y, початкову точку на двовимірної площині.

Клас панелі p4sLine буде містити лінію екрану для малювання, змінну *error* с тестом помилки і делегат - обробник *Paint* :

```
class p4sLine: Form
{
sLine sl; // лінія екрану для малювання
public p4sLine (sLine l) { mkPan (l);
}
protected void mkPan (sLine l) { sl = l;
if (l == null) {
error = "no any line!"; Paint + = _paint;
}
else if (l.ps == null || l.ps.Length <1) {
error = String.Format ( "Line '{0}' is empty!", l.nm); Paint + = _paint;
}
else {// тут додати делегат-обробник для малювання лінії
}
BackColor = Color.Ivory;
Dock = DockStyle.Fill;
} // повідомлення про помилку
public void _paint (object sender, PaintEventArgs e) {
// віртуальна площину? на якій малюють Graphics g = e.Graphics;
// шрифт
using (Font f = new Font ( "Times New Roman", 14))
                              { g.DrawString // намалювати
                              (Error // цей текст
                              , F // цим шрифтом
}
```

```
                                , Brushes.Red // червоною пензликом
                                , 10, 3 * 14 // у таких координатах x і y
                    );
                }
            }
        }
```

Ордината Y на віртуальній площині збільшується від верхнього краю вікна до нижнього. Конструктор викликає спеціальний метод ініціювання панелі mkPan, тому що в майбутньому його доведеться викликати з конструктора-спадкоємця після (а не до) дій по ініціалізації об'єкта-спадкоємця.

- Клас лінії екрану (sLine) поки порожній, точніше, містить раніше обговорений метод getPoints і назву лінії:

```
class sLine { public string nm = "line"; }
```

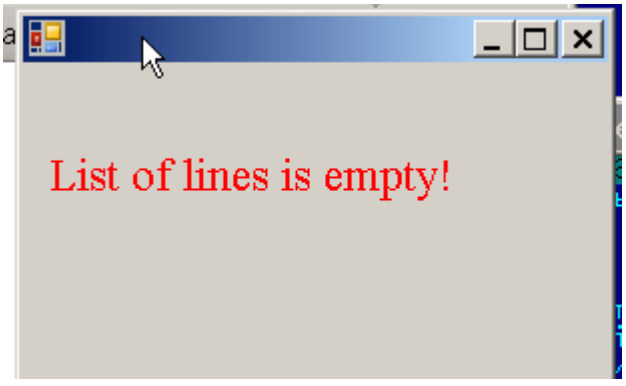
- Клас вікна w4line містить єдиний керуючий елемент - панель p4sLine - для малювання лінії екрану.

```
class w4line: Form
{
p4sLine pan;
public w4line (sLine ln) {
Padding = new Padding (10);
pan = new p4sLine (ln);
Controls.Add (pan);
}
}
```

- Статичний метод Main класу Program:

```
static void Main ()
{
sLine ln = new sLine (); ln.nm = "test line";
Application.Run (new w4line (ln)); // показали лінію
}
```

побудує і покаже вікно w4line з повідомленням про помилку



Мал. Виведення тексту в режимі графіки

```
}
```

• Додати в клас sLine - лінії екрану масив точок для малювання - ps, за замовчуванням олівець – pen та дві ознаки: малювати незамкнений набір ліній (трек) - mkLine або малювати замкнутий полігон - mkPolygon.

```
class sLine {
public string nm = "line";
public Point [] ps; //Координати точок лінії в пікселях
public Pen pen = Pens.Black; //Олівець для основного малювання
public bool mkPolygon = false;
public bool mkLine = true;
}
```

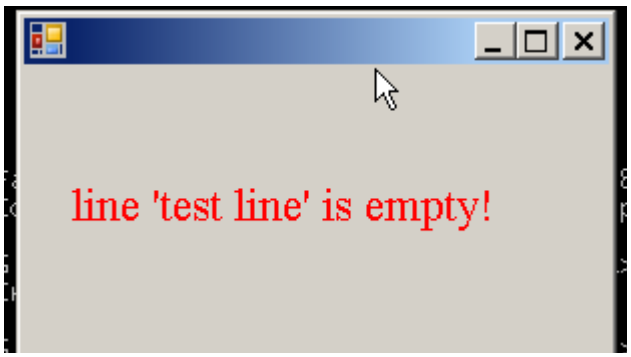
• Додати в клас sLine делегат - обробник події Paint, який повідомляє про помилку, при відсутності точок в лінії:

```
public void _paint (object sender, PaintEventArgs e) { // малювання лінії
Graphics g = e.Graphics;
 if (ps! = null && ps.Length> 1)
  { if (mkPolygon)
     g.DrawPolygon (pen, ps); // замкнена лінія
    else
     g.DrawLines (pen, ps);
  }
else {
string text = String.Format ( "line '{0}' is empty!", nm);
using (Font f = new Font ( "Times New Roman", 14)) {
        g.DrawString (text, f, Brushes.Red, 20, 3 * 14); }
   }
}
```

• Змінити в класі p4sLine конструктор, додавши в останню гілку делегат-обробник (підписати метод на подію) sl.paint в події Paint:

```
else {// додати делегат-обробник для малювання лінії
sl = l;
Paint + = sl._paint; // делегат - обробник лінії
}
```

Повідомлення про помилку змінюється на таке:



}

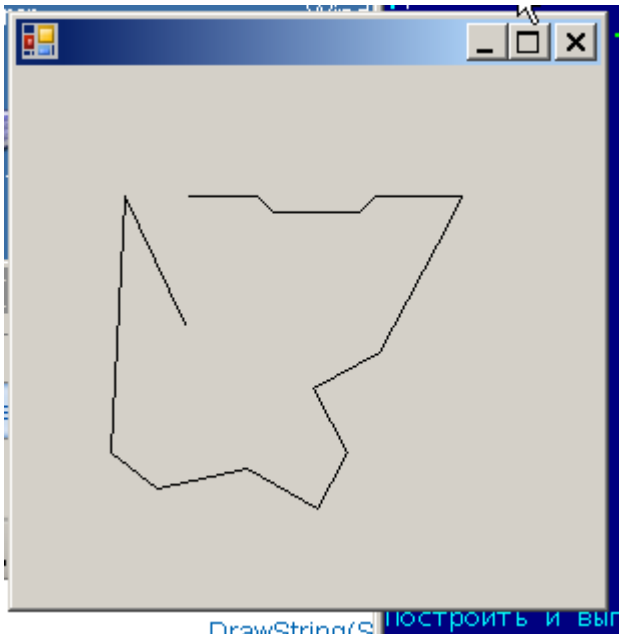
• Розробити і виконати в Main метод getPoints

```
ln.getPoints (Console.In); // прочитали лінію
// для чого набрати файл 1.txt такого змісту:
86 64
120 64
128 72
171 72
179 64
222 64
210 88
181 142
148 160
165 192
150 220
115 200
70 210
47 192
54 64
84 128
```

• Побудувати і виконати додаток:

```
Program.exe <1.txt
```

Додаток виводить вікно:



Приклад ламаної лінії у вікні

• Самостійно додати можливість підписувати точки ламаної, використовуючи метод DrawString, розглянутий вище.

```
}
```

4.3 ЛАБОРАТОРНА РОБОТА: МАСШТАБУВАННЯ ЛАМАНИХ

Зміст заняття: Основи програмування графіки.

Створити додаток для масштабування кривої в головному вікні програми, передбачити можливість масштабування та переміщення ламаної на екрані за допомогою кнопок панелі інструментів ToolBar.

- Повторити використання класів ToolBar, ToolBarButton (або ToolStrip);
- Використовувати поле ToolBarButton.ToolTipText для демонстрації підказки при попаданні покажчика мишки в область панелі інструментів.
- Мати два масиви: перший (coors) масив пар дійсних чисел повинен використовуватися для зберігання реальних координат кривої (наприклад градуси світової системи координат 1984 року), другий (pxls) - масив пар цілих чисел, який буде зберігати координати пікселів у вікні для відображення кривої.
- Познайомитися з методом Invalidate.
- Познайомитися зі способами проектування площині на площину.
- Створити клас mapping з обов'язковим методом для відображення точок реальності в точки екрану

```
void map (double X, double Y // точки реальності ,
            Out int x, out int y // точки екрану
) {} ,
```

з методами для зміни параметрів відображення, наприклад:

```
public void moveNordSouth ( int shift /// <відсотки зсуву області видимості,> 0 - на північ, <0 - на південь ) {}
public void moveNordSouth ( int shift /// <відсотки зсуву області видимості,> 0 - на північ, <0 - на південь ) {}
public void zoom ( int shift
 // <> 0 --збільшення області видимості (дрібний масштаб)
) {}
```

та методом-обробником події натискання на кнопки панелі інструментів, який має змінювати параметри класу mapping, за масивом реальних координат будувати масив координат пікселів екрану та викликати метод Invalidate для області вікна, в якій виконується малювання.

Додаток з лабораторної роботи 4.2 виконує побудову та відтворення немасштабованної ламаної. Для того, щоб масштабувати цю ламану, додаток потрібно доопрацювати, щоб делегати-обробники події Paint виконувати дії лише з малювання ліній. Ця вимога призводить до необхідності виконувати перетворення реальних даних (наприклад, лінія типу rLine, яка зберігає дані в градусах, для кожного числа типу double повинна мати 6 або 8 розрядів після коми) в дані, готові для відображення методами .Net, наприклад, DrawPolygon. Таким чином, кожне натискання на кнопки буде приводити до зміни параметрів відображення mapping, що спричиняє необхідність до перерахунку лінії екрану, а потім, до виклику методу Invalidate, який, у свою

```
}
```

чергу, призводить до події Paint. Крім цього, відображення mapping сильно пов’язано з розмірами панелі, на якій відбувається малювання лінії. Це визначає необхідність створювати клас-спадкоємець керуючого елемента Panel, який буде містити клас для відображення і обслуговутиме дані лінії екрану sLine.

Таким чином, стає зрозумілим, для чого потрібно створювати клас для пари дійсних чисел - tuple2d, клас для зберігання лінії з даними з реальності - rLine, клас для перетворення точок лінії реальності в пікселі лінії екрану - mapping (він має виконувати основну функціональність), клас p4sLine2 (спадкоємець p4sLine), який буде займатися відображенням всієї лінії реальності в лінію екрану після зміни масштабу або положення центру вікна.

Корисна інформація при розробці програми для лабораторної роботи.

- Скопіювати в новий каталог з каталогу лабораторної малювання немасштабованної прямой усі файли;
- Додати файл tuple.cs:

```
class tuple2d  {// <точка реальності в двовимірному просторі
public double X; // longitude east
public double Y; // latitude nord
public tuple2d (double x = 0.0, double y = 0.0) { X = x;  Y = y;  }
public tuple2d (string x, string y) { X = double.Parse (x);
Y = double.Parse (y); } }
```

який буде містити пари координат реальної ламаної.

- Додати файл rLine.cs з областю видимості System.Threading:

```
class rLine {/// <лінія реальності
public tuple2d [] pnts; // <Координати точок лінії в пікселях public string nm;
public static readonly char NumberDecimalSeparator =
Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator
[0];
public rLine (string nm, string nordNm = "", string eastNm = "") {
 this.nm = nm;  }
public static tuple2d [] = // читання масиву точок
 getPoints (TextReader tr, bool vFlag = false, char decPnt = ’.’)
{   return null;  }
```

де властивість NumberDecimalSeparator потрібна для подальшої правильної обробки даних, в яких дійсні числа можуть містити десяткову точку або десяткову кому, в залежності від налаштувань на комп’ютері.

- Вміст методу Main поміняти на наступне:

```
rLine ln = new rLine ( "test real line");
ln.pnts = rLine.getPoints (Console.In, true); // прочитали лінію
Application.Run (new w4line (ln)); // показали її
```

- Повністю змінити клас w4line:

```
class w4line: Form
  }
```

```
{  rLine rl;
   public w4line (rLine l) { rl = l;
   Padding = new Padding (10);  }
}
```

Тепер додаток компілюється і показує порожнє вікно.

- Створити файл p4sLine2.cs з класом спадкоємцем класу p4sLine:

```
class p4sLine2: p4sLine
{  public p4sLine2 (rLine l)  {  }
}
```

- Додати в клас вікна керуючий елемент «смугу інструментів» і панель для малювання:

```
ToolBar tb = new ToolBar (); p4sLine2 pan;
```

У конструкторі вікна до смуги інструментів додати кнопки, створити панель і поки не викликати делегат обробник натискання на смугу інструментів:

```
public w4rLine (rLine l)
{  rl = l;
   Padding = new Padding (10); AutoSize = true;
   StartPosition  =  FormStartPosition.CenterScreen;
   pan = new p4sLine2 (rl); Controls.Add (pan);
tb.ButtonSize = new System.Drawing.Size ((int) (200/3), (int) (40));
ToolBarButton Zoom_p = new ToolBarButton ( "Zoom +");
ToolBarButton  Zoom_m = new ToolBarButton ( "Zoom -");
ToolBarButton Left = new ToolBarButton ( "Left");
ToolBarButton Right = new ToolBarButton ( "Right");
ToolBarButton Up = new ToolBarButton ( "Up");
ToolBarButton Down = new ToolBarButton ( "Down");
tb.Buttons.AddRange (new ToolBarButton []
{Zoom_p, Zoom_m, Left, Right, Up, Down}); tb.Dock = DockStyle.Top;
// tb.ButtonClick + = ToolBarButtonClick; !!!!! делегат поки за коментовано
Controls.Add (tb);
```

- Поки має вийти не надто красиво (рис.4.3.1)

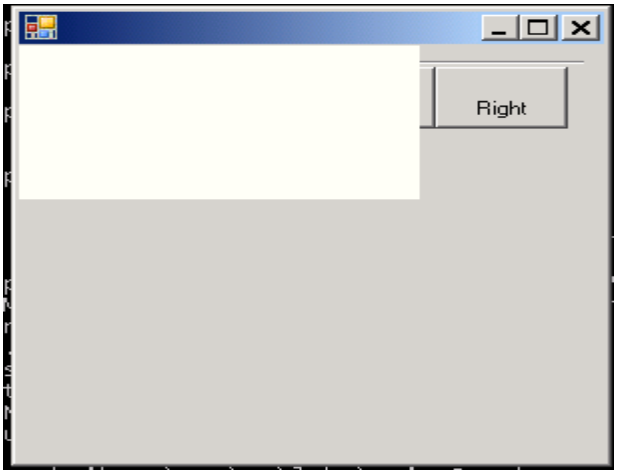


Рис.4.3.1. Побудоване вікно зі смугою інструментів

```
}
```

- Створити файл mapping:

```
public class mapping
{
int l, r, // розміри вікна
t, b; // b верх, t низ
bool isWellformed = false;
public double xMin, xMax, yMin, yMax;
// кордону ламаної в реальності
// змінив ці значення можна домогтися ефекту зміщення
// або зміни масштабу
public mapping (): this (0.0, 1.0, 0.0, 1.0) {
this.l = 0;
this.b = 0;
this.r = 255;
this.t = 255;
}
public mapping (double x, double X, double y, double Y)
{
xMin = x; xMax = X; yMin = y; yMax = Y;
if (xMin == xMax) // числа повинні бути різними щоб уникнути поділу на
нуль xMax = xMin + 0.01;
if (yMin == yMax) yMax = yMin + 0.01;
this.l = 1;
this.b = 1;
this.r = 400;
this.t = 300;
isWellformed = true;
}
public int w {
get {return r + l;} // поле розміру l зліва і справа
}
public int h {
get {return t + b;} // поле розміру b зверху і знизу
}
}
```

де поля xMin, xMax, yMin, yMax гратимуть важливу роль у перетворенні ламаної з реального вигляду в пікселі вікна. Вони задають координати кутів мінімального прямокутника, що містить вихідну лінію.

- Тепер можна доопрацювати вікно панель для ламаної p4sLine2:

```
class p4sLine2: p4sLine
{
public mapping mp;
// панель буде містити відображення для масштабування і зсуву
    }
```

```
public p4sLine2 (rLine l)
{
mp = new mapping ();
// оператор не зможе змінювати розмір вікна
MinimumSize = new Size (Convert.ToInt32 (mp.w), Convert.ToInt32 (mp.h));
MaximumSize = new Size (Convert.ToInt32 (mp.w), Convert.ToInt32 (mp.h));
AutoSize = true; if (l == null) ;
else
 if (l.pnts == null || l.pnts.Length <1)
       { sl = new sLine (); // в цьому випадку
         sl.nm = l.nm; // ще нічого малювати
       }
 else {
 sl = new sLine ();
 sl.nm = l.nm; // вже можна формувати лінію екрану
 // <--- наприклад, в цьому місці
 Paint + = sl._paint; // делегат - обробник лінії
 }
mkPan (sl);
}
}
```

Тепер вікно має прийняти пристойний вигляд (рис.4.3.2):

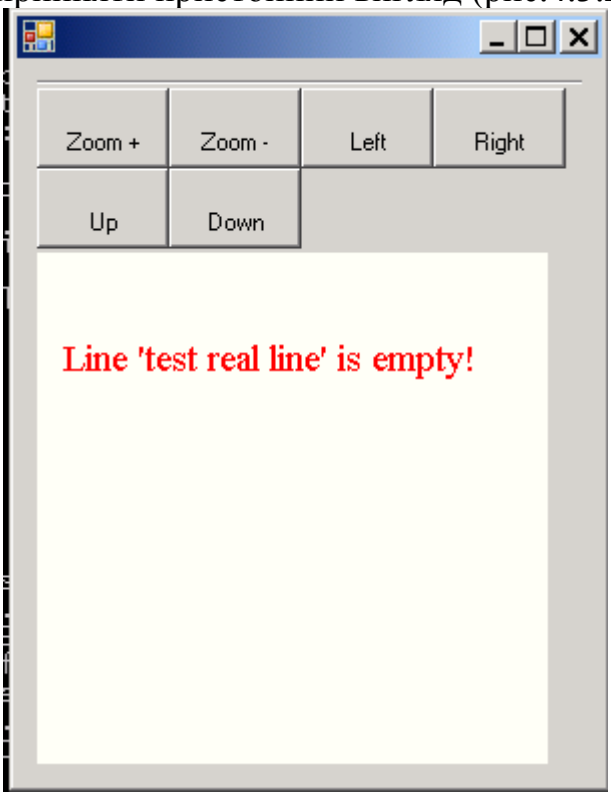


Рис.4.3.2. Нове вікно зі смугою інструментів

• Можна починати читання даних. Метод getPoints при читанні цілих не повинен представляти труднощі, але потрібно не забувати замінювати заданий символ десяткового дробу на символ, прийнятий в системі:

```
}
```

```
public static tuple2d [] // читання масиву точок
getPoints (TextReader tr, bool vFlag = false, char decPnt = '.')
{
List <tuple2d> ps = new List <tuple2d> ();
....
for (; (r = tr.ReadLine ())! = null; lineNo ++) {
 if (NumberDecimalSeparator! = decPnt)
 r = r.Replace (decPnt, NumberDecimalSeparator);
....
}
...
return ps.ToArray ();
}
```

Дані зчитано, їх треба використати їх при створенні панелі p4sLine2.

- З прочитаних даних про лінію отримати крайні (ліві, праві, верхні, нижні) значення обох координат:

```
class rLine {/// <лінія реальності
public bool getBox (out double xMin, out double xMax, out double yMin, out
double yMax) {
xMin = double.MaxValue;
xMax = double.MinValue;
yMin = double.MaxValue;
yMax = double.MinValue;
if (pnts! = null && pnts.Length> 0)
 { for (int i = 0; i <pnts.Length; i ++) {
    if (pnts [i] .X <xMin) xMin = pnts [i] .X;
    if (pnts [i] .X> xMax) xMax = pnts [i] .X;
    if (pnts [i] .Y <yMin) yMin = pnts [i] .Y;
    if (pnts [i] .Y> yMax) yMax = pnts [i] .Y;
 }
return true; }
return false; } }
```

- Використаємо отримані дані для створення панелі p4sLine2 у процесі малювання ламаної:

```
public p4sLine2 (rLine l)
{ double a, b, c, d;
   if (l.getBox (out a, out b, out c, out d))
   { mp = new mapping (a, b, c, d);
   // mp.mkZmEqual (); // це метод для того, щоб масштаб по обох вісях
   // був однаковий
 ...
 }
else
mp = new mapping (); // цей рядок вже був записаний раніше

    }
```

Тепер може бути побудований додаток, наприклад, за командою a.exe:

a.exe <1.txt

де 1.txt - файл з лабораторної про малювання немасштабованої ламаної. В результаті маємо отримати вікно розміром 400 на 300 пікселів (рис.4.3.3):

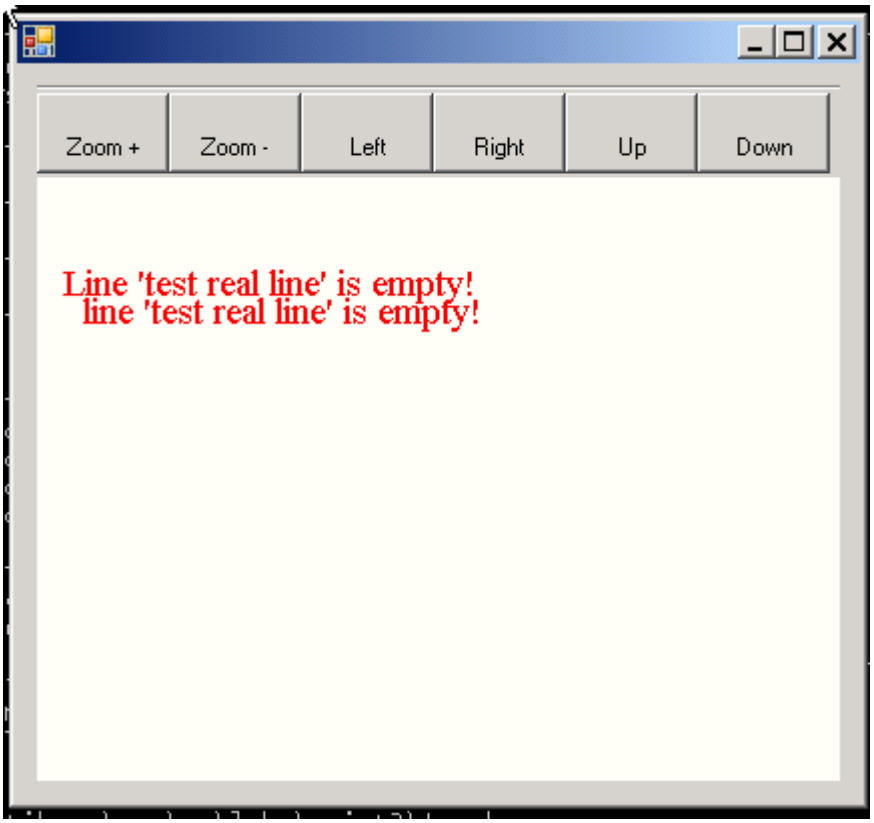


Рис.4.3.3. Вікно (400x300) зі смугою інструментів

Можна приступати до малювання ламаної.

• В клас p4sLine2 додати метод mkSLine для створення лінії на екрані з лінії реальності:

```
public void
mkSLine (rLine rl) { int x, y;
sl.ps = new Point [rl.pnts.Length];
for (int i = 0; i <rl.pnts.Length; i ++) {
mp.map (rl.pnts [i] .X, rl.pnts [i] .Y  // вхідні координати
, Out x, out y);                        // вихідні координати на екрані
sl.ps [i] = new Point (x, mp.h - y);    // додати їх в лінію і перевернути
} }
```

Зауваження. mp.h-y створює ефект перевертання ламаної, який потрібно виконати внаслідок того, що лівий верхній кут панелі для малювання має координату (0,0) і значення ординати збільшуються вниз.

• Застосувати метод mkSLine в конструкторі панелі p4sLine2:

```
sl = new sLine (); sl.nm = l.nm;
mkSLine (l); /// <--- застосування
Paint + = sl._paint; // делегат - обробник лінії

}
```

• В клас відображення mapping додати, власне, саме відображення:

```
public
void map (double X, double Y, out int x, out int y) {
x = Convert.ToInt32 (Math.Round ((X - xMin) / (xMax - xMin) * (r-l) + l));
y = Convert.ToInt32 (Math.Round ((Y - yMin) / (yMax - yMin) * (t-b) + b)); }
else { x = 0; y = 0; }
}
```

де дане відображення виводиться з пропорцій:

(X - xMin) / (xMax - xMin) = (x - l) / (r-l)
(Y - yMin) / (yMax - yMin) = (y - b) / (t-b)

Змінюючи xMax і xMin можна домогтися зміни масштабу або зсуву по осі зліва - направо.

Тепер лінію екрану має бути видно (рис.4.3.4):

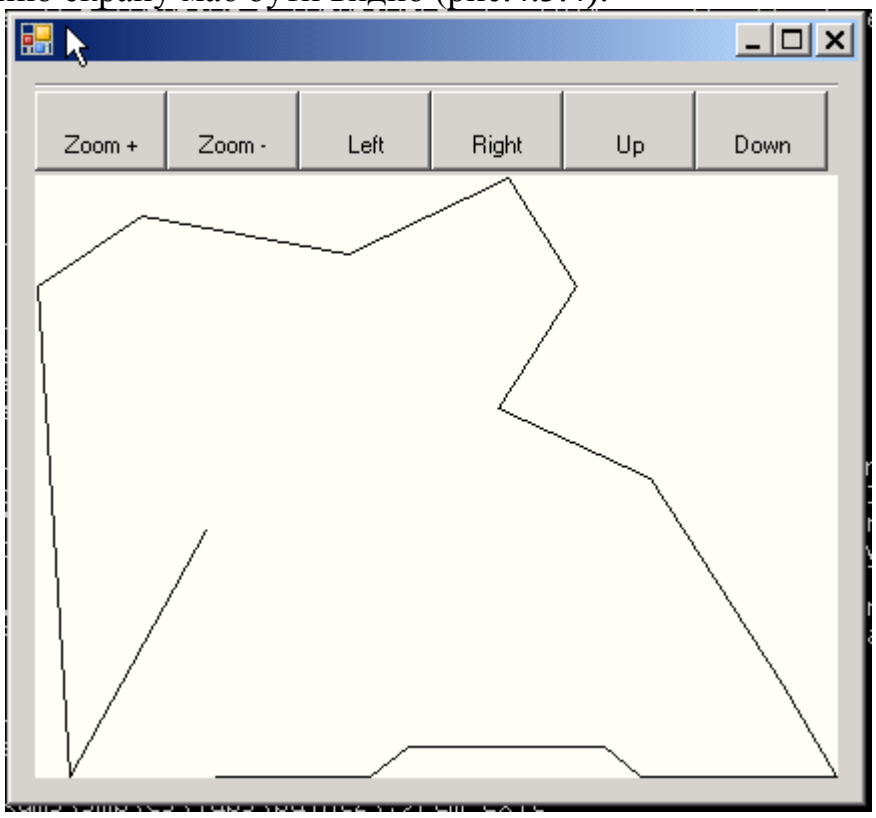


Рис.4.3.4. Ламана лінія

• В клас відображення mapping додати два методи дозволяють виконувати зсув по осях зліва - направо і зверху - вниз:

```
public void moveNordSouth (
int shift) // <відсотки зсуву області видимості,> 0 - на північ, <0 - на південь
 {
  if (isWellformed) {
  double percent = (yMax - yMin) / 100; yMax + = percent * shift;
  yMin + = percent * shift;
  }
 }

    }
```

```
public void moveEastWest (
int shift) // <відсотки зсуву області видимості,> 0 - на північ, <0 - на південь
 {
  if (isWellformed) {
  double percent = (xMax - xMin) / 100; xMax + = percent * shift;
  xMin + = percent * shift;
 }
}
```

• У вікно w4line додати делегат-обробник натискання кнопок на панелі інструментів ToolBarButtonClick:

```
void ToolBarButtonClick (object sender, ToolBarButtonClickEventArgs e)
{
 if (e.Button.Text == "Zoom +") // поміняти параметри відображення
; // pan.mp.zoom (-10);
else if (e.Button.Text == "Zoom -")
; // pan.mp.zoom (10);
else if (e.Button.Text == "Right") pan.mp.moveEastWest (20);
else if (e.Button.Text == "Left") pan.mp.moveEastWest (-20);
else if (e.Button.Text == "Down") pan.mp.moveNordSouth (-20);
else if (e.Button.Text == "Up") pan.mp.moveNordSouth (20);
else
return;
pan.mkSLine (rl); // перерахувати лінію екрану
pan.Invalidate ();  // викликати подію Paint
}
```

І, остаточно, розкоментувати його застосування в конструкторі вікна:

```
tb.ButtonClick + = ToolBarButtonClick; !!!!! делегат закоментований
```

Додаток має реагувати на натискання кнопок ліворуч, праворуч, вгору, вниз. Зрозуміло, що для руху необхідно, щоб обидва граничних значення однієї з осей координат змінювалися в одну сторону. Наприклад, в методі moveNordSouth:

```
yMax + = percent * shift; yMin + = percent * shift;
```

Це означає, що для зміни масштабу zoom треба менші границі зменшувати, а більші границі - збільшувати, на зразок цього:

```
yMax + = percentY * shift; yMin- = percentY * shift; xMax + = percentX * shift;
xMin- = percentX * shift;
```

Залишилося додати метод mkZmEqual, щоб масштаби по осях вниз - вгору і зліва - направо були однаковими.

```
}
```

4.4 ЛАБОРАТОРНА РОБОТА: ВИКОРИСТАННЯ МИШІ

Зміст заняття: Основи програмування графіки. Обробка подій.

Створити додаток для масштабування ламаної лінії в головному вікні програми, передбачити можливість масштабування та зміни ламаної на екрані за допомогою кліків мишки. Передбачити можливість повідомляти оператору чи потрапив клік мишки по області вікна поруч з ламаної.

- Для обчислення відстані і напрямку переміщення графіка використовувати події маніпулятора «миша»:

* MouseDown - почати переміщення і відзначити початкову точку для обчислення напрямку і відстані переміщення;

* MouseUp - припинити перетягування;

* MouseMove - викликати ефект перетягування кривої. Для кожної такої події визначати поточну точку для обчислення напрямку і відстані переміщення, перераховувати новий масив координат точок в пікселях і викликати метод Invalidate () для відтворення поточного становища ламаної.

- Познайомитися з класом ContextMenuStrip.

- Познайомитися з класом Region і GraphicsPath.

- При попаданні маркером миші в область вікна поруч з намальованою ламаною створити спливаюче контекстне меню з деяким змістом, при попаданні маркером миші в область, яка розташована далеко від ламаної, створити контекстне меню з іншим змістом.

Корисна інформація при розробці програми для лабораторної роботи.

• Вступ до засобів роботи з маніпулятором «миша».

Події (тип - делегат), що будуть оброблятися в додатку для роботи з мишею: натисканняя клавіші, переміщення покажчика, відпускання клавіші. Їм відповідають делегати класу Form: mouseDown, MouseMove, MouseUp. В аргументі делегата-обробника типу MouseEventArgs необхідно задати поле, що містить координати покажчика миші - Location. Функціональність для роботи з мишею буде додаватися в код спадкоємця класу w4rLine.

У ньому потрібно відкрити доступ до полів - реальна лінія rl типу rLine і панель малювання лінії *pan* типу p4sLine2.

Дописати до них ключове слово protected:

```
protected rLine rl; protected p4sLine2 pan;
```

• Створити клас msWnd, спадкоємець w4rLine:

```
class msWnd: w4rLine
{
public msWnd (rLine l): base (l) {}
}
```

• В клас msWnd додати дві змінні:

```
Boolean mouseDown; // запам’ятали, що клавіша мишки натиснута
PointF MousePoint1; // координати мишки в момент натискання клавіші
```

• Дописати два делегата-обробника натискань на клавіші мишки:

```
}
```

```
void Form1_MouseDown (object sender, MouseEventArgs e)
{ MousePoint1 = e.Location;
mouseDown = true; // клавіша натиснута
}
Form1_MouseUp (object sender, MouseEventArgs e)
{ mouseDown = false; // клавіша відпущена
}
```

- У конструкторі вікна додати їх до відповідних подій:

```
public msWnd (rLine l): base (l) { mouseDown = false;
pan.MouseDown + = Form1_MouseDown; pan.MouseUp + = Form1_MouseUp;
}
```

- Додати обробник для перерахунку лінії при зсуві лінії:

```
void Form1_MouseMove (object sender, MouseEventArgs e)
{ if (mouseDown) {
PointF MousePoint2 = e.Location; // поточне положення вказівника
pan.mp.moveEastWest ((Int) (// вирахувати відсоток від зсуву по екрану
(MousePoint1.X - MousePoint2.X) * 100) /ClientSize.Width);
pan.mp.moveNordSouth // тут перевернута система координат
(- (((int) (MousePoint1.Y - MousePoint2.Y)) * 100) /ClientSize.Height);
pan.mkSLine (rl); // при великих обчислювальних витратах pan.Invalidate ();
// ці рядки можна перемістити в Form1_MouseUp MousePoint1 = e.Location;
// лінію перерахували, у слід. раз перерахувати щодо цієї точки
}
}
```

- Контекстне меню

Для створення керуючих елементів контекстного меню використовуютьсяє два набору класів: перший - ContextMenu і MenuItem; другий - ContextMenuStrip і ToolStripMenuItem. Меню створювати при натисканні на ліву кнопку миші (клік):

```
// MouseClick + = Control1_MouseClick; // десь в конструкторі вікна void
Control1_MouseClick (Object sender, MouseEventArgs e)
{ Region r = mkRgn (); // побудувати регіон,
// тобто, безліч пікселів, поруч з ламаної
if (r.IsVisible (Cursor.Position))
// потрапила точка в регіон?
{ ContextMenuStrip.Items.Clear (); // видалити елементи меню
  ToolStripMenuItem mi // створити новий елемент меню
  = New ToolStripMenuItem ( "in Region");
  ContextMenuStrip.Items.Add (mi); // додати його в контекстне меню
  mi.Click + =ContextMenuStrip_Click; // додати його обробник
}
else
  }
```

```
{ ContextMenuStrip.Items.Clear  ();
  ToolStripMenuItem mi = new ToolStripMenuItem ( "Not in Region");
  ContextMenuStrip.Items.Add (mi);
  mi.Click + = ContextMenuStrip_Click;
}
}
void ContextMenuStrip_Click (object sender, EventArgs e)
{ Console.WriteLine ( "MenuIten have been clicked"); }
```

- Об’єкт типу Region (внутрішня частина геометричної фігури, яка складається з прямокутників і контурів) має різні методи для перевірки попадання деякого прямокутника (або точки) всередину регіону - IsVisible. Застосування методу наведено вище. Регіон створюється з складеного контуру (об’єкта типу GraphicsPath), який будується послідовним додаванням деяких багатокутників, що описуються біля кожного ребра ламаної. Наприклад, так:

```
Point [] mkPolygon (Point a, Point b)
{ Point [] m = new Point [5]; m [0] = a;
 m [1] = b;
 m [2] // трохи відійшли від точки b
 = New Point (b.X + 4, b.Y);
 m [3] // трохи не дійшли до точки a
 = New Point (a.X + 4, a.Y);
 m [4] = a; // замкнули багатокутник return m;
}
```

Тоді побудова шуканого регіону mkRgn може виглядати так:

```
Region mkRgn ()
{
GraphicsPath gp = new GraphicsPath ();
for (int i = 0; i <points.Length - 1; i ++) {
        gp.AddPolygon (mkPolygon (points [i], Points [i + 1]));
}
        return new Region (gp);
}
```

Завершений варіант програми має дозволяти зміну контекстне меню, після кожного кліка лівої клавішею, в залежності від попадання покажчика миші в область поряд з ламаної.

```
}
```

4.5 ЛАБОРАТОРНА РОБОТА: ЗАВАНТАЖЕННЯ ФАЙЛІВ З МЕРЕЖІ

Зміст заняття: Основи мережевого програмування.

Створити додаток для скачування файлів по заданою адресою в інтернеті (wget).

- Познайомитися з класом HttpWebRequest.
- Познайомитися з класом HttpWebResponse.

Корисна інформація при розробці програми для лабораторної роботи.

Знайомство з протоколами, що використовуються в комп'ютерних мережах, можна починати з класів:

HttpWebRequest (його батьківський клас WebRequest), який виконує запит до уніфікованого ідентифікатором ресурсу (англ. URI Uniform Resource Identifier);

HttpWebResponse - надає відповідь від уніфікованого ідентифікатора ресурсу;

WebException - використовується для опису виняткових ситуацій, що виникають при появі помилок під час доступу додатки до комп'ютерної мережі;

HttpStatusCode – перерахування, яке містить значення кодів станів, визначених для протоколу HTTP.

Для створення нових об'єктів типу HttpWebRequest, потрібно використовувати статичний метод Create батьківського класу WebRequest. Далі, за допомогою методу GetResponse необхідно отримати об'єкт класу HttpWebResponse і, потім, за допомогою метора GetResponseStream отримати об'єкт класу Stream. Останній об'єкт надає низькорівневий доступ до байтів, що повертає уніфікований ідентифікатор ресурсів.

Наведемо приклад скачування початкової сторінки пошукача Google.

```
using System;
using System.Data; using System.Threading; using System.Net;
using System.IO;
class application1
{
[STAThread]
static int Main(string[] ars)
{ int ch ;
  byte []b =new byte[1]; Stream  istrm=null;
  Stream so = Console.OpenStandardOutput();
{ HttpWebRequest req=(HttpWebRequest) WebRequest.Create
  ("http://google.com.ua");
 HttpWebResponse resp = (HttpWebResponse) req.GetResponse();
 istrm  = resp.GetResponseStream();
 while (-1 != (ch=istrm.ReadByte()) )
   { b[0]=(byte)ch; so.Write(b, 0, 1); }
}
```

```
 }
return 0;
}
}
```

4.6 ЛАБОРАТОРНА РОБОТА: ЖУРНАЛЮВАННЯ РОБОТИ ДОДАТКУ

Зміст заняття: Потоки команд і процеси.

Створити бібліотеку для журналювання роботи програми. Продемонструвати її роботу на прикладі багатопоточного обчислення простих чисел.

- Познайомитися з класом Thread.
- Познайомитися з класом Monitor.
- Познайомитися з класом Interlocked.
- Познайомитися з інструкцією lock.

Для виконання лабораторної роботи потрібно гарантувати потрапляння рядків в файл журналу в тому порядку, в якому потоки зверталися до методу виведення повідомлень в журнал (аналог методу WriteLine);

Корисна інформація при розробці програми для лабораторної роботи.

Для розв’язання задачі викостаємо чергу повідомлень. Черга Queue - це клас з простору колекцій System.Collections. Її особливість полягає в тому, що методи додавання в чергу - Enqueue та видалення з черги - Dequeue працюють за правилом - "Перший прийшов, перший пішов". Це правило означає, що елементи черги встановлюються в ній у тій же послідовності, в якій вони до неї заносилися. Метод для постановки повідомлень в чергу буде викликатися з нитки (процесу) користувача, а для видалення повідомлень з черги і виведення їх в файл буде використовуватися спеціальна нитка, яка працює з низьким пріоритетом. Вона буде вибирати повідомлення та виводити їх за наяної можливості, тобто, у той час, коли не працюватимуть призначені для користувача процеси. Крім того, повідомлення прийнято сортувати з урахуванням важливості, що дозволяє гнучко керувати кількістю виведених повідомлень. Отже, потрібно послідовно виконати:

- створити перерахування на кшталт такого:

```
public enum IMPORTANCELEVEL {
Debug // налагоджувальні повідомлення
, Warning // попередження - не важливо повідомлення
, Error // помилки програми - важливі повідомлення,
, FatalError // катастрофічні помилки,
// роблять неможливість функціонування додатку
};
```

- створити клас журналу:

```
public class Logger: IDisposable

}
```

```
{
 bool dbg = false; // закривати-відкривати файл після кожного виведення
 StreamWriter sw = null; // потік виводу повідомлення
 Queue StringQueue = null; // черга повідомлень
// Встановлено рівень важливості (повідомлення нижче встановленого рівня ігноруються)
// рівень важливості треба задавати в конструкторі, при створенні журналу
IMPORTANCELEVEL ImportanceLevel = IMPORTANCELEVEL.Error;
// Потік виведення повідомлень в файл Thread Log;
// Bool для виходу з нескінченного циклу в потоці логера
 bool Working = false;
}
```

Зрозуміло, що наслідування від IDisposable, пов'язане з файлом виводу StreamWriter sw та інструкцією using. Ці особливості обговорювалися раніше.

- створити метод, який займається висновком повідомлень в файл:

```
void LogMessage () { string p = null;
// Нескінченний цикл виведення повідомлень раз в секунду
while (Working) {
// Виведення всіх повідомлень, якщо є
 lock (this) {
  if (StringQueue.Count> 0) {
  p = (string) StringQueue.Dequeue ();
  sw.WriteLine (p);
// в дуже проблемних випадках доводиться закривати файл sw після
// виведення кожного повідомлення, щоб побачити повідомлення, які
// програма не встигла скинути в файл перед аварійним завершенням
 if (dbg) {
  sw.Close ();
  sw = new StreamWriter (filename, true); }
 p = null;
}
else { Thread.Sleep (10); //якщо повідомлень немає, то можна затримати цикл
    }
   }
  }
}
```

- Конструктор Logger-а може містити код, наприклад, такого вигляду:

```
 this.StringQueue = new Queue ();
 this.sw = new StreamWriter ( "logger.txt", true);
 this.working = true;
 this.Log = new Thread (new System.Threading.ThreadStart (LogMessage));
 this.Log.Priority = ThreadPriority.Lowest;
 this.Log.Start ();

   }
```

Процеси можуть бути основні та фонові (властивість IsBackground). Метод Main чекатиме завершення основних процесів і завершує роботу фонових (при цьому не гарантується виконання операторів з секцій finally фонових ниток).

- створити метод для завершення роботи та звільнення файлу:

```
public void Dispose () {
 string p = null;
// Намагаємося зупинити логер this.Working = false;
 if (Log! = null) Log.Join ();
// потім виводимо залишок черзі в цьому методі
while (StringQueue.Count> 0)   { // Якщо є ще не виведені повідомлення,
 p = (string) StringQueue.Dequeue (); // !!!!
 sw.WriteLine (p);
 p = null; }
sw.Close ();
}
```

Працювати буде тільки поточний процес, тому черга не блокується інструкцією lock;

- залишилося написати метод для журналювання повідомлень:

```
void WriteLine (
IMPORTANCELEVEL Importance // важливість даного повідомлення
, String Format // рядок з форматом повідомлення
, Params object [] Segments // параметри повідомлення
) {
// Перевірка рівня важливості повідомлення
if (Working && Importance> = this.ImportanceLevel) {
// форматування повідомлення
String Message = String.Format ( "[{0}] \ t", Importance) + "\ t" + string.Format
(Format, Segments);
// додавання повідомлення в чергу
lock (this) { StringQueue.Enqueue  (Message); }
 }
}
```

На цьому роботу можна закінчити. Бажано, щоб ім’я журналу не співпадало з logger.txt, а будувалося з назви програми. Наприклад, якщо додаток називається app.exe, то журнал повинен мати назву app.txt. Крім цього, для реалізації можна використовувати не клас Queue, а шаблон черги Queue <T>:

```
Queue <string> StringQueue = new Queue <string> ();
```

що дозволить не турбуватися про перетворенні типу та бути впевненим у відсутності помилок при видаленні об’єкта з черги (дивись, наприклад, метод Dispose):

```
p = StringQueue.Dequeue (); // !!!!
```

```
}
```

Таким чином, залишилося написати додаток мінімум з двох потоків, який протестує використання розробленого класу журналювання. Наведемо для зразку приклад виконання операції інкремента в двох різних процесах з коментарями і описом способів синхронізації різних ниток.

У цьому додатку виконується операція інкремента по одній змінній у двох процесах які по черзі виконуються, з синхронізацією ниток.

За допомогою умовної компіляції можна синхронізувати потоки (в даному прикладі це гарантоване збільшення значення змінної обома потоками по черзі) за допомогою декількох способів: за допомогою класу Interlocked, за допомогою класу Monitor, який полегшує використання так званої критичної секції і за допомогою двох способів використання інструкції lock.

```
// #define INTERLOCKED
// #define MONITOR
//#define LOCK
#define  LOCKSTATIC
using System; using System.IO;
using System.Threading; namespace test {
class m    {
static public int   max = 5; // пять секунд работать
static public int    sNumbers = 0; ///< количество целых чисел, проверенных средом
public int numbers = 0;
[STAThread]
public static void Main()
{
m someMemory   = new m(); // спільна пам’ять для обох потоків
p b = new p("ThreadPriority.Lowest " , someMemory);
p a = new p("ThreadPriority.Highest", someMemory,
                ThreadPriority.Highest);
b.t.Join();   // очикуємо завершення потоку b
a.t.Join();  // очикуємо завершення потоку a
string s;
#if INTERLOCKED
s = "interlocked";
#elif  MONITOR
s = "monitor   ";
#elif  LOCK
s = "lock      ";
#elif  LOCKSTATIC
s = "lockStatic ";
#else
#error  You have to set at least one macro
#endif

}
```

```
Console.WriteLine("\n--{0} version, total(nonstatic/static): {1}/{2}\n",
 s, someMemory.numbers,  m.sNumbers);
Console.WriteLine("thread '{0}' finished with {1} numbers",
 a.name, a.numbers);
Console.WriteLine("thread '{0}' finished with {1} numbers",
 b.name, b.numbers);
Console.WriteLine("\n--{0}version, difference(nonstatic/static): {1}/{2}\n",
 s,  someMemory.numbers -  a.numbers - b.numbers,
 m.sNumbers -  a.numbers - b.numbers );
 }
}

class p  {
public Thread t     = null;
public int   numbers = 0;
public string name  = null;
public p (string o, m  sharedMemory,
    ThreadPriority p=ThreadPriority.Lowest){ name = o;
        t = new Thread(work);
        t.Start(sharedMemory);     // передали спільну пам'ять в потік
}
public  void work(object o){
m mm = (m) o;               // отримали спільну пам'ять в потоці
DateTime st = DateTime.Now;
for (DateTime cur = DateTime.Now;
(cur - st).TotalSeconds < m.max;
cur = DateTime.Now) {
#if INTERLOCKED
 Interlocked.Increment(ref  mm.numbers);
#elif  MONITOR
 Monitor.Enter (mm); mm.numbers ++; Monitor.Exit (mm);
#elif  LOCK
 lock(mm) { // змінна mm типу клас, а не структура
 mm.numbers++  ;
}
#elif  LOCKSTATIC
 lock(typeof(m)) {
 m.sNumbers++;    // це спільну змінна
#endif
}
numbers++;    // змінна середовища виконання для того, щоб дізнатися
}                  // який потік більше відпрацював
}
}
}

}
```

4.7 ЛАБОРАТОРНА РОБОТА: РОЗРОБКА ДОДАТКУ ДЛЯ РОБОТИ З РОЗПОДІЛЕНОЮ БАЗОЮ ДАНИХ

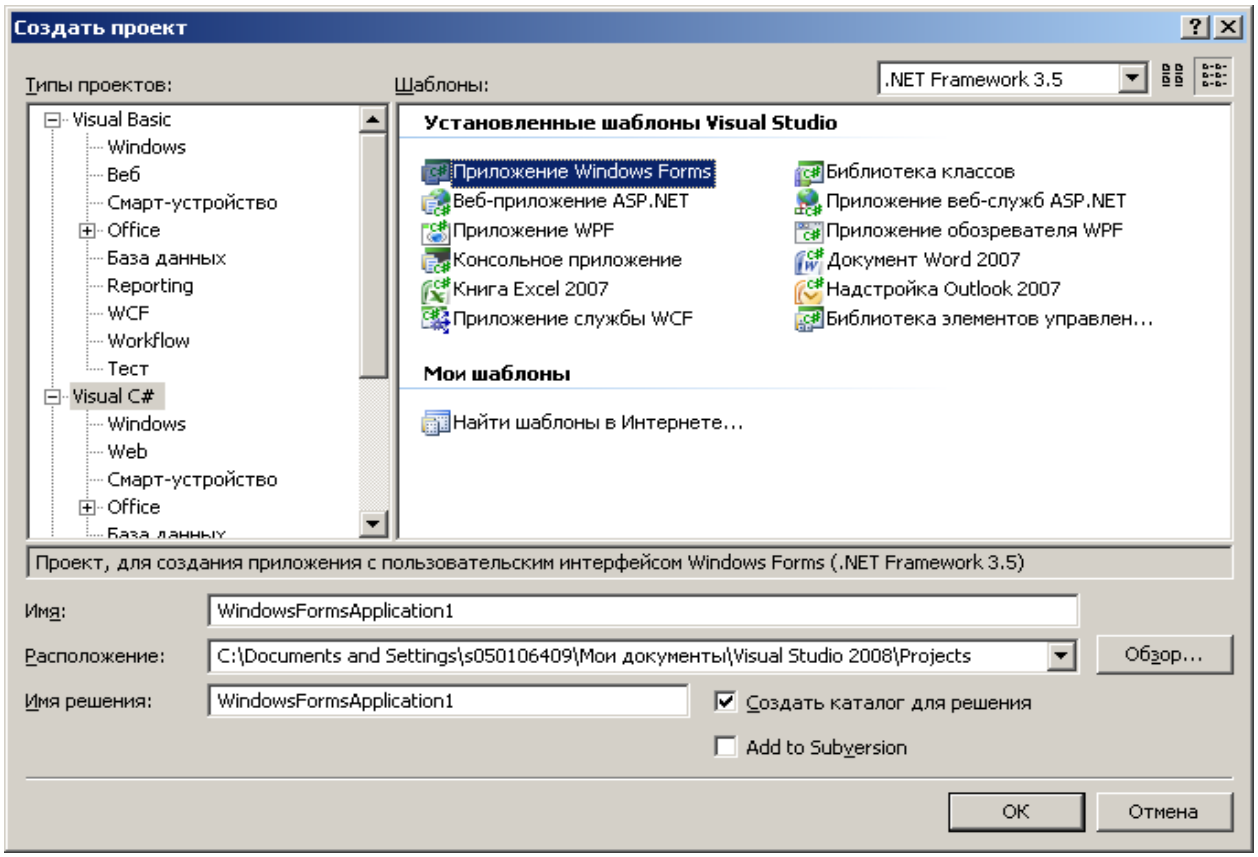
Зміст заняття: Отримати навички створення Windows-додатків з графічним інтерфейсом для роботи з розподіленою базою даних.

Створити проект Windows Forms на C# в рамках середовища розробки Visual Studio.

В процесі виконання лабораторної роботи потрібно забезпечити доступ до СУБД PostgreSQL. Для цього необхідно встановити драйвер Npgsql (скачати за посиланням [32]). Бібліотеки Mono.Security.dll, Npgsql.dll треба розмістити у папці з проектом. У проекті необхідно вказати посилання на ці бібліотеки.

Корисна інформація при розробці програми для лабораторної роботи.

Для створення нового проекту C# у середовищі розробки Visual Studio необхідно обрати в меню **Файл→Создать→Проект**. У списку типів проектів вибрати мову C# і пункт «**Приложение Windows Forms**» (мал.4.7.1).



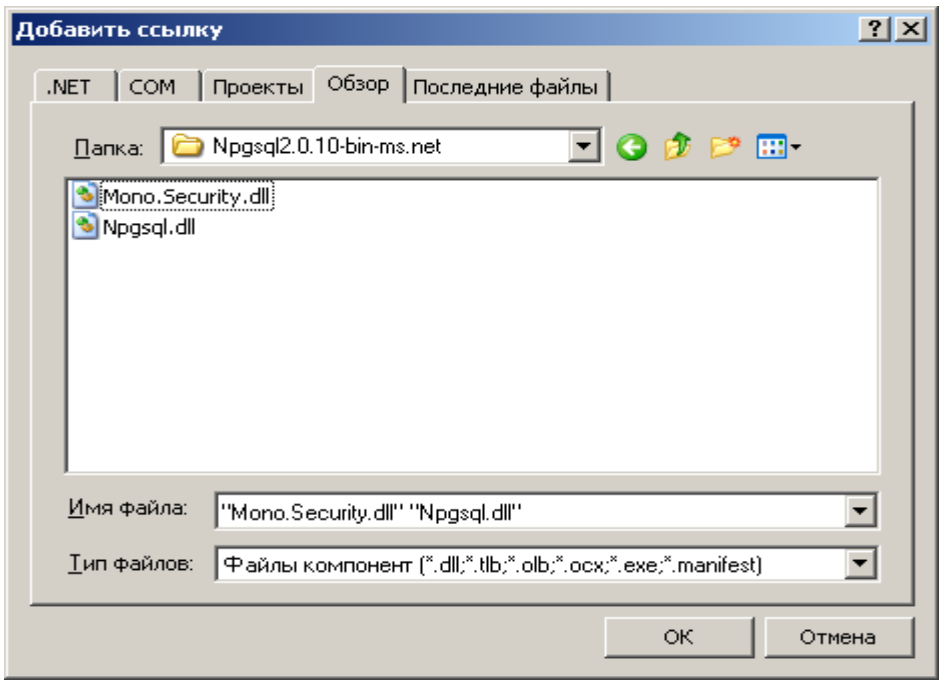
Мал. 4.7.1. Створення проекту

Для забезпечення підключення до бази даних:

- в огляді рішень правою кнопкою маніпулятора миші на меню Посилання вибрати пункт Додати посилання. На вкладці Огляд необхідно вказати шлях для библиотек (див. мал. 4.7.2). В коді форми за допомогою правої кнопки миші обрати ім’я форми (за замовчанням **Form1.cs**) та вибрати **Перейти до коду.** Далі необхідно прописати рядок

using Npgsql;

```
}
```

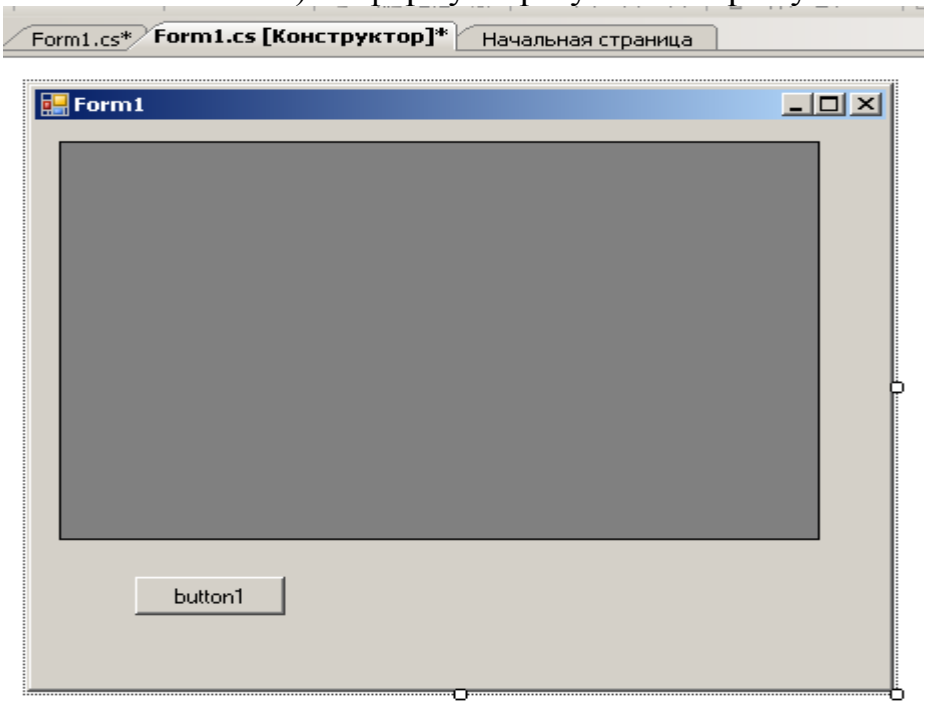


Мал.4.7.2. Додавання бібліотек у проект

• Для створення нового підключення до бази даних, прописати рядок підключення в довільному місці програми перед зверненням до бази даних. Рядок підключення виглядає так:

NpgsqlConnection conn = new NpgsqlConnection ("server=<Сервер>; database=<база данных>; user Id=<Имя пользователя>; password=<Пароль>");

• Для забезпечення ви**воду** вибірки даних використаємо **е**лемент **DataGridView**. Цей елемент розташовано на панелі елементів (за замовчанням викликається **Ctrl+Alt+X**) в розділі **Дані** (див. мал. 4.7.3). Необхідно розмістити цей елемент, а також кнопку (**Button** з розділу стандартні панелі елементов) на форму. В результаті отримуємо таку форму



Мал. 4.7.3. Додавання DataGridView і кнопки на форму

}

• Щоб створити обробник для кнопки, необхідно зробити подвійне натискання по кнопці, після чого автоматично відбувається перехід до коду форми:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

При натисканні кнопки повинен відбуватися вивод даниxx в **DataGridView**. Для цього в обробник натискання кнопки слід прописати такий код:

```
// Створюємо новий набір даних
DataSet datasetmain = new DataSet();
// Відкриваємо підключення
conn.Open();
// Очищуємо набір даних
datasetmain.Clear();
// Sql-запит
NpgsqlCommand command = new NpgsqlCommand("Текст запроса",
conn);
// Новий адаптер потрібен для заповнення набору даних
NpgsqlDataAdapter da = new NpgsqlDataAdapter(command);
// Заповнюємо набір даних даними, які повернув запит
da.Fill(datasetmain, "table1");
// Зв’язуємо елемент DataGridView1 з набором даних
dataGridView1.DataSource = datasetmain;
dataGridView1.DataMember = "table1";
// Закриваємо підключення
conn.Close();
```

Форма готова.

Зауваження. Додавання даних, редагування та видалення пропонується розробити на лабораторних заняттях або самостійно.

4.8 ЛАБОРАТОРНА РОБОТА: РОБОТА З MICROSOFT WORD

Зміст заняття: Отримати навички створення Windows-додатків з графічним інтерфейсом для роботи з текстовим процесором Microsoft Word.

Створити проект Windows Forms на C# в рамках середовища розробки Visual Studio.

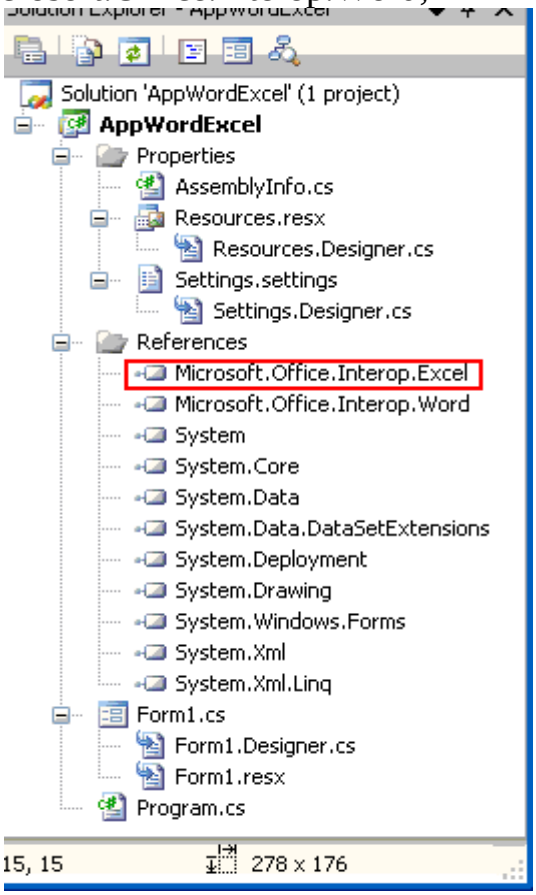
В процесі виконання лабораторної роботи потрібно продемонструвати работу з документами, параграфами, символами, закладками и т.і. Завдання виконується шляхом застосування та обробки властивостей та методів цих об’єктів. Об’єкти при створенні рішень потрібно визначати глобально для того, щоб забезпечити доступ до них з довільної функції проекта.

Корисна інформація при розробці програми для лабораторної роботи.

```
}
```

• Для роботи з Microsoft Word необхідно додати посилання на відповідні бібліотеки (див. мал. 4.8.1). В коді програми потрібно написати:

using Word = Microsoft.Office.Interop.Word;



Мал.4.8.1. Додавання бібліотек

З точки зору додатку, всі об'єкти Word мають ієрархічну структуру. Об'єкт **Application** – це COM-сервер і оболонка для інших об'єктів. Він може містити один або декілька об'єктів **Document**. Об'єкти **Document** можуть містити такі об'єкти, як **Paragraph, Table, Range, Bookmark, Chapter, Word, Sentence, Sections, Headers, Footers**,… і т.і. Точніше кажучи, об'єкт **Application** може містити колекцію **Documents** – посилань на об'єки типу **Document**, а кожний об'єкт типу **Document** – колекцію **Paragraphs** або посилань на об'єкти типу **Paragraph** і т.д.

• Створення документу. Зафіксуємо глобальне визначення основного об'єкту Word.Application:

```
private Word.Application wordapp;
```

Створимо нові об'єкти Word, параграф і документ:

```
// Створюємо об'єкт Word. Це відповідає запуску Word
wordapp = new Word.Application();
// Зробимо його видимим
wordapp.Visible = true;
// Створимо об'єкт параграф
Word.Paragraph wordparagraph;

}
```

```
Word.Document doc = new Word.Document();
// Створюємо документ
doc = app.Documents.Add (ref object Template, ref object NewTemplate,
ref object DocumentType, ref object Visible);
```

Параметры метода Add:

Template – ім’я шаблона, за яким створюється новий документ. Якщо значення не вказано, то використовується шаблон Normal.dot;

NewTemplate – при значенні true новий документ відкривається як шаблон. Значение за замовчанням – false;

DocumentType – тип документу, може прийнати одне з наступних значень констант типа Word.WdNewDocumentType:

wdNewBlankDocument – документ Word (за замовчанням);
wdNewEmailMessage – электронне повідомлення;
wdNewWebPage – Web-сторінка;
wdNewXMLDocument – XML-документ;

Visible – видимість документу. При значенні true (за замовчанням) документ відображається.

В якості параметра Template методу Add можно призначити ім’я існуючого документа або повне ім’я шаблону.

- Вивод тексту виконується не просто у параграф, а у діапазон параграфу – об’єкт **Range.**

Об’єкт **Range** – це неперервна область документу, що включає позиції початкового та кінцевого символів. Для порожнього параграфу або випадку спів падіння позицій діапазону, **Range** представляє курсор ввода.

В документі можна визначити діапазон, викликаючи метод **Range з** передачею йому початкового та кінцевого значень позицій символів (при визначенні позиції номеру символів рахують від 0 і розглядають всі символи, в тому числі й недруковані). Виділений діапазон можна "підсвітити", використовуючи метод **Select()**:

```
Object begin = 0; Object end = 5;
Word.Range wordrange = worddocument.Range(ref begin, ref end);
wordrange.Select();
```

- Створення таблиці проводиться на основі інформації про об’єкти. Інформація про об’єкти **Table** зберігається у вигляді посилань на таблиці документа у властивості **Tables**. Набір посилань **Tables** доступний з об’єктів **Document**, **Selection** та **Range,** також з об’єкта **Word.Table**. Це означає, що і створювати таблиці можна з використанням довільного з цих об’єктів. Для створення таблиць використовується метод **Add**:

```
Add (
Word.Range Range, // Об’єект Range - місце формування таблиці
int NumRows,     // Кількість рядків
int NumColumns,  // Кількість стовбців

}
```

```
/*  Визначаємо,  чи змінює Word автоматично розміри комірок в таблицях,
щоб вони відповідали змісту комірок.
Може бути одна з Word.WdDefaultTableBehavior констант:
wdWord8TableBehavior (ні) або wdWord9TableBehavior (так).
За замолчанням - wdWord8TableBehavior.
*/
ref object DefaultTableBehavior,
/*
Автопідбір ширини стовбців,
одна з наступних Word.WdAutoFitBehavior констант:
wdAutoFitContent – за змістом, wdAutoFitFixed - фиксована або
wdAutoFitWindow – за шириною вікна.
Если DefaultTableBehavior встановлено в wdWord8TableBehavior, цей
параметр ігнорируєтся.
*/
ref object AutoFitBehavior
)
```

Детальніше з роботою з Microsoft Word в C# можна ознайомитися в [33].

4.9 ЗАВДАННЯ ДЛЯ МОДУЛЬНОЇ РОБОТИ

1. Утиліта для демонстрації літаючого кола, коло повинне відштовхуватися від меж поля.
2. Утиліта для демонстрації двох різних контекстних меню, в залежності від того, був клік виконаний всередині багатокутника (зображеного у вікні) або зовні.
3. Утиліта для зафарбовування двох трикутників в залежності від їх взаємного розташування:
   - трикутники не перетинаються - зафарбовувати одним кольором.
   - трикутники мають одну спільну сторону (або спільну точку) - зафарбовувати різними кольорами.
   - трикутники перетинаються - НЕ зафарбовувати взагалі.
4. Віконна утиліта для запуску машини Тьюринга [3]. Програма має графічно відображати переміщення головки машини Тьюринга на стрічці.
5. Віконна утиліта для реалізації машини Тьюринга Lavrinovi4:TM. Додаток може не висвітлювати переміщення головки машини Тьюринга на стрічці, але має використовувати домен додатків.
6. Утиліта для збору висот місцевості на заданій ділянці.
7. Утиліта для збору температури на заданій ділянці.
8. Утиліта для збору курсу валют.
9. Додаток для перегляду і редагування бази даних Постачальників з журналюванням роботи програми [8].
10. Практичне застосування правил оформлення та роботи з документами Microsoft Word.

```
}
```

ДЕЯКІ ПРАКТИЧНІ АСПЕКТИ СУЧАСНОГО ЗАСТОСУВАННЯ МОВИ C#.

Використання платформи розробки додатків Docker

Docker — це відкрита платформа для розробки, доставки і експлуатації додатків. Docker розроблений для швидшого створення та поширення розроблених додатків. За допомогою Docker можна відокремити кожен додаток від конкретної системної архітектури та поводитися з програмною інфраструктурою як з керованим додатком. Docker допомагає викладати розроблені додатки швидше, швидше їх тестувати і зменшити час між написанням коду і запуску коду. Docker робить це на основі платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають управляти і викладати створені додатки.

У своєму ядрі Docker дозволяє запускати практично будь-який додаток, безпечно ізольований в контейнері. Безпечна ізоляція дозволяє запускати на одному хост-комп'ютері декілька контейнерів одночасно. Інтегрована природа контейнера, який запускається без додаткового навантаження супервізора, дозволяє розробникам добиватися більше від наявного архітектурного та програмного забезпечення.

Платформа та засоби контейнерної віртуалізації можуть бути корисними в випадках:

- упаковування додатку та раніше використовуваних компонент в docker контейнери;
- доставка цих контейнерів вашим командам для розробки та тестування;
- викладання цих контейнерів у дата-центрах і в хмарах.

Головні компоненти Docker

Платформа Docker складається з двох головних компонент:

- Docker-платформа віртуалізації з відкритим кодом;
- Docker Hub: платформа-as-service для розповсюдження і управління Docker контейнерами.

Архітектура Docker

Docker використовує архітектуру клієнт-сервер. Docker клієнт спілкується з демоном Docker, який бере на себе підтримку створення, запуску, розподілу контейнерів.

Docker-демон

Програма-демон за пускається на хост-машине. Користувач не взаємодіє з сервером на пряму, а використовує для цього Docker-клієнт.

Docker-клієнт

Docker-клієнт, програма Docker — головний інтерфейс до Docker. Вона отримує команди від користувача і взаємодіє з Docker-демоном.

Внутрішні компоненти Docker-а

- образи (images)
- реєстр (registries)
- контейнери

}

Образи

Docker-образ — це шаблон. Наприклад, образ може містити операційну систему на основі Linux з Apache і додатком на ній. Образи використовуються для створення контейнерів. Docker дозволяє легко створювати нові образи, оновлювати ті, що вже існують або використовувати образи, створені іншими розробниками. Образи — це компоненти зборки Docker-a.

Реєстр

Docker-реєстр зберігає образи. Існують публічні і приватні реєстри, з яких або на які можна завантажити образи. Публічний Docker-реєстр — це Docker Hub. Там зберігається величезна колекція образів. Реєстри — це компоненти розповсюдження.

Контейнери

Контейнери структурно схожі на директорії. У контейнерах міститься все, що потрібне для роботи додатку. Кожен контейнер створюється з образу. Контейнери можуть бути розроблені, запущені, зупинені, перенесені або видалені. Кожен контейнер ізольований і є безпечною платформою для функціонування додатку. Контейнери — це компоненти роботи.

Простір імен (namespaces)

Docker використовує технологію namespaces для організації ізольованих робочих просторів, що називаються контейнерами. Коли запускається контейнер, Docker створює набір просторів імен для даного контейнера. Це дозволяє створити ізольований рівень, де кожен аспект контейнера запущений у своєму просторі імен і не має доступу до зовнішньої системи.

Список деяких просторів імен, які використовує Docker:

- **pid:** для ізоляції процесу;
- **net:** для управління мережевими інтерфейсами;
- **ipc:** для управління IPC ресурсами. (ICP: InterProccess Communication);
- **mnt:** для управління точками монтування;
- **utc:** для ізолювання ядра і контролю генерації версій (UTC: Unix timesharing system).

Control groups (контрольні групи)

Docker також використовує технологію ***cgroups*** або контрольні групи. Їх застосування – це ключ до роботи додатку в ізольованому вигляді, надання додатку лише тих ресурсів, яких він потребує. Цей підхід гарантує, що контейнери будуть коректно поводити себе в конкретних умовах обчислювального середовища. Контрольні групи дозволяють розділяти доступні ресурси апаратних компонентів і, якщо необхідно, встановлювати межі та обмеження . Наприклад, обмежити обсяг пам'яті контейнеру.

Union File System

Union File Sysem або UnionFS — це файлова система, яка працює створюючи рівні, роблячи процес формування контейнерів дуже швидким. Платформа Docker використовує UnionFS для створення блоків, з яких будується контейнер. Docker використовує декілька варіантів UnionFS, включаючи: AUFS, btrfs, vfs і DeviceMapper.

}

Формати контейнерів

Платформа Docker поєднує компоненти в обгортку, яку називають форматом контейнера. Формат, що використовується за замовчанням, називається *libcontainer*. Платформа Docker підтримує також традиційний формат контейнерів в Linux з допомогою LXC.

Основні компоненти Docker-проекту для створення образів

Розробка додатку .NET Core передбачає застосування середовища **mono** або **coreclr**. Послідовність етапів зборки:

- Створення порожнього .NET Core проекту.
- Підготовка файлу project.json, який має містити такі поля:

```
"dependencies": {
   "Microsoft.AspNet.IISPlatformHandler": "1.0.0-rc1-final",
   "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final"
  },
  "commands": {
   "web": "Microsoft.AspNet.Server.Kestrel --server.urls http://*:5004"
  } , ...
```

- Перевірка запуску додатку локально

```
dnvm list
dnu restore
dnx web
```

- Запуск браузера та перехід за адресою http://localhost:5004. Якщо є привітання, проект готовий до пакування в контейнер.
- Підготовка файлу Dockerfile в корені проекту.

```
FROM microsoft/aspnet:1.0.0-rc1-update1-coreclr
COPY . /app
WORKDIR /app
RUN ["dnu", "restore"]
EXPOSE 5004
ENTRYPOINT ["dnx", "-p", "project.json", "web"]
```

- Зборка образу:

```
docker build -t first-aspnet-app .
```

- Запуск образу:

```
docker run -d -p 80:5004 first-aspnet-app
```

Приклад розробки тестового додатка в Docker наведено, наприклад, за адресою https://habr.com/ru/company/microsoft/blog/278173/.

}

ПИТАННЯ ДЛЯ ІСПИТУ

Питання за темою Створення віконних додатків

1. Які властивості класу Form використовуються для створення багатодокументного інтерфейсу?
2. Напишіть метод для вибору та активізації вже відкритого дочірнього вікна.
3. Який клас використовується для створення вікон?
4. Які методи використовуються для демонстрації вікон?
5. які властивості пов’язані з кнопками управління вікном?
6. Які властивості і перерахування використовуються при створення діалогових вікон?
7. Перерахуйте і охарактеризуйте стандартні діалогові вікна.
8. Як event delegate використовується при розробці віконного інтерфейсу?
9. Яку подію потрібно використовувати перед закриттям вікна? Наведіть приклад обробника.
10. Які класи, свойства і методи використовуються для завантаження - збереження XML документа?
11. Які класи, свойства і методи використовуються для вилучення вузлів з XML документа?
12. Які класи, свойства і методи використовуються для додавання вузла в інший XML вузол?
13. Які класи, свойства і методи використовуються для додавання атрибута до XML вузлу?
14. Назвіть кілька керуючих елементів. Як керуючий елемент додається до вікна?
15. Яке ключове слово призначене для створення методу зі змінним числом параметрів? Наведіть приклад.
16. Назвіть керуючі елементи, призначені для автоматичного розміщення керуючих елементів елементів що містяться в них.
17. Наведіть приклад використання шаблону для створення словника.
18. Для чого використовується властивість Padding?
19. Для чого можна використовувати керуючий елемент Panel?
20. Яка властивість і перерахування використовується для розміщення керуючих елементів?
21. Які класи використовуються для перегляду табличних даних?
22. Наведіть приклад налаштування керуючого елемента для перегляду табличних даних.
23. Наведіть приклад додавання в керуючий елемента для перегляду табличних даних трьох колонок.
24. Яка колекція і метод використовується для додавання рядків в таблицю?
25. Як гарантувати видимість заданої осередку в керуючому

}

елементі для перегляду табличних даних DataGridView?

26. Які властивості, класи, методи і події застосовуються для використання головного меню у вікні?

27. Які класи і властивості застосовуються для використання панелі інструментів у вікні?

28. Наведіть приклад інформаційного керуючого елемента, відмінного від Label.

29. За допомогою якого ключового слова гарантується звільнення захоплених програмних ресурсів (маються на увазі файли, нитки (Потоки команд), сокети і з'єднання з базою даних)? Наведіть приклад.

30. Який клас використовується для створення (або читання, або виведення) текстів в заданій кодуванні? Приклад створення і використання кодування 1251.

Питання за темою Додаткових можливостей програмування у .Net

1. Які засоби існують в .Net для визначення типу під час виконання?
2. Яка утиліта використовується для генерації ресурсів?
3. Який клас використовується для генерації ресурсів?
4. За допомогою якого класу здійснюється доступ до ресурсів?
5. Які методи використовуються для перетворення цілих чисел і рядків (string) в колір (Color)?
6. Назвіть і охарактеризуйте основні класи, які використовуються для малювання в вікнах .Net.
7. Створіть червоний Карадаш за допомогою словесного опису кольору.
8. Створіть напівпрозорий синій пензлик.
9. Який метод панелі слід викликати, що б перемалювати панель?
10. Наведіть приклад обробника події Paint для малювання ламаної.
11. Як створюються спливаючі підказки на елементах управління (наприклад, на ToolBarButton)?
12. Яким методом вимірюють розмір тексту в пікселях? Наведіть приклад.
13. Яким методом виводять текст в області вікна (без об'єкта класу Label)? Які додаткові класи для цього требуются?
14. Який клас використовується для переміщення відображаються у вікні об'єктів у вертикальному напрямку? Наведіть приклад ініціалізації.
15. Який клас використовується для переміщення відображених у вікні об'єктів в горизонтальному напрямку? Наведіть приклад ініціалізації.
16. В якому класі містяться звичайні математичні функції? Наведіть приклад.
17. Який клас використовується для низькорівневого (Побайтного) введення - виведення? Наведіть приклад.
18. Який клас використовується для виконання Http - запитів? Наведіть приклад скачування файлу.

}

19. Як можна дізнатися положення координат покажчика мишки 1) на екрані, 2) у вікні?

20. Які класи, властивості і методи використовуються для створення контекстного меню?

21. Які класи, властивості і методи використовуються для зміни (додавання, видалення елементів) контекстного меню?

22. Наведіть приклад створення контекстного меню.

23. Якими засобами можна перевірити чи належить задана точка складної геометричної фігури?

24. Що таке процес і який клас використовується для управління виконуваними на комп'ютері процесами і запуску нових (дочірніх) процесів?

25. Що таке код завершення програми? Яким способом додаток може задати код завершення (або код повернення)?

26. Що таке потік команд (нитка)? Який клас використовується для запуску та управління нитками?

27. Який клас використовується для організації виняткового доступу ниток до, наприклад, цілим змінним?

28. Що таке блок синхронізації і які класи і методи призначені для роботи з ними.

29. Для чого використовується оператор lock? Наведіть приклад його застосування.

30. Як організувати винятковий доступ ниток до статичних змінних?

Для того, щоб організувати виключний доступ ниток до статичних змінних в поєднанні з методами Monitor.Enter (object o) / Monitor.Exit (object o) або оператором lock необхідно використовувати операцію typeof.

}

ТЕМАТИКА КУРСОВИХ ПРОЕКТІВ

При оцінки курсових проектів заохочуються роботи, виконані з урахуванням раніше створених проектів таких як

- Бібліотеки обробки аргументів командного рядка, див. [9].
- Журналювання додатків, див. [14], [21].
- Бібліотеки геометричних обчислень на еліпсоїд див. [13].
- або отдокументування за допомогою системи Doxygen (див. [10]).

Звіти до курсових робіт, викладеним в цьому розділі повинні задовольняти вимогам до оформлення звітів розділу 9 і додатково містити дві UML - діаграми. Різні приклади UML - діаграм, що відносяться до водних потоків даних, можна подивитися в роботі [15, с.~4] або [15, с.~7]. Діаграми будувалися за допомогою вільно-розповсюджуємої утиліти graphviz (Див. [11]). Приклади UML - діаграм класів, які описує структуру системи, що демонструє класи системи, їх атрибути, методи і залежності між класами, можна подивитися в пояснювальній записці до проекта [9] (файл args.2.10. pdf). Ця діаграма був згенерована з коду програми за допомогою утиліти doxygen (див. [10]).

Крім того, результати використання системи Doxygen можна подивитися в методичних рекомендаціях до лабораторної роботи Масштабована ламана (див. п.4.3).

Список тем курсових проектів:

1. Утиліта для асинхронного читання великих файлів - заповнення керуючого елемента для показу таблиць (DataGridView).
2. Гра ’Теніс’ для двох гравців.
3. Гра ’П’ятнашки’.
4. Клас для роботи з GPX - файлами, сумісний з Бібліотекою обчислень на еліпсоїд elGeo, див. [13].
5. Утиліта для злиття - поділу GPX - файлів.
6. утиліта для знаходження зупинок на треку транспортного засобу.
7. Утиліта для знаходження кутів оцифрованих карт Системи Координат 1942 року.
8. Додаток для відображення треків транспортного засобу на картах Системи Координат 1942 року, [4]).
9. Утиліта для відображення треків транспортного засобу на картах відкритих сервісів, (Див. [24], [5]).
10. Ускладнена версія останніх двох додатків з демонстрацією кадрів з записів відеореєстратора, [6].
11. Утиліта для знаходження знака ’Стоп’ на кадрі відеореєстратора.
12. Утиліта для виявлення лінії горизонту на кадрах відеореєстратора.
13. Додаток для отримання і відтворення потоку радіотрансляції і демонстрації поточного часу.

}

14. Віконна утиліта для примусового сортування (і транслітерацією) музичних файлів в дереві каталогу.

15. Утиліта для збору висот для заданої точки, лінії або прямокутника з відкритих географічних сервісів, [25].

16. Утиліта для збору курсів валют.

17. Утиліта для збору температури.

18. утиліта для вивантаження опису географічних об’єктів з файлів проекту OSM в CSV - файли.

19. утиліта для вивантаження координат географічних об’єктів з файлів проекту OSM в CSV - файли.

20. Додаток для обертання тривимірної фігурки.

21. Документування модульної роботи за допомогою кроссплатформенної системи документування початкового програмного коду - Doxygen (http://www.doxygen.nl/), введення можна подивитися в [10].

22. Виконання модульної роботи за допомогою системи контролю версій Git на хостингу https://github.com/.

23. Клас для за рахунками площі Сферосидичного багатокутника і додаток для тестування класу.

24. Додаток для навчання дітей письму. БД повинна містити таблиці Слів, Картинок (кожному слову - кілька різних картинок), Букв (с картинками зображень букви, з яких дитина буде складати слово), Повідомлень Додатки (як реакція програми на дії дитини), Спроби (історію спроб дітей).

25. Додаток розподілу часу лекцій, лабораторних і самостійної роботи студента для побудови робочої навчальної програми.

26. Додаток для демонстрації порушення Принципу підстановки Лісків (див. [12]), яке виникає при спробах наслідування дійсних чисел від цілих або навпаки.

27. Утиліта для стресового тестування файлової системи.

28. Багатодокументне застосування по опису системи обмеження доступу з [16], замість БД використовувати текстові CSV файли.

29. Утиліта для вивантаження інформації з файлів MS Excel.

30. Проходження відбіркового турніру будь-якого програмістки конкурсу на http://codeforces.ru.

ВИМОГИ ДО ОФОРМЛЕННЯ ЗВІТІВ З ВИКОНАНИХ РОБІТ

Титульна сторінка повинна містити назву міністерства, університету, інституту, кафедри, тему роботи, виконавця, викладача, місто і рік.

Обсяг роботи не менше 10 аркушів. Текст треба набирати шрифтом Times New Roman, 12 pt, інтервал між рядками - 1, форматування по ширині, відступи: 2 см зліва, 1.5 - справа, зверху і знизу сторінки, абзац - 1 см. Нумерація сторінок - внизу по центру.

```
}
```

Список літератури оформляти згідно вимог [2].

Звіти до робіт повинні містити наступні частини:

- титульний лист;
- зміст;
- постановка задачі;
- теоретична частина (Опис предметної області, короткі теоретичні відомості, необхідні для виконання завдання, опис інструментів програмування і т.д.);
- відповіді на питання відповідного модуля. Відповіді на питання повинні містити номер питання, його формулювання і файл з фотографією тексту, написаного від руки;
- опис алгоритму програми;
- опис використання програми;
- тести для перевірки працездатності;
- висновки;
- список використаної літератури.

У звітах до лабораторних деякі частини дозволяється пропускати. Не дозволяється пропускати розділи «Теоретична частина» та «Відповіді на питання». Теоретична частина для лабораторних формується у накопичувальній формі, тобто, в звіті до поточної лабораторної дописується лише те, що треба для її розробки та не описується те, що було наведено в звітах до попередніх лабораторних. Теоретична частина для модуля повинна містити всі питання з поточного модуля, пов’язані з додатком. Розділи «Відповіді на питання» і «Теоретична частина» повинні містити обсяг інформації, достатній для безпосереднього використання, у відповіді можна додавати лише інформацію, яка стосується тематики питання. Наприклад, відповідь на питання: ’Цілі типи’, повинна містити список усіх цілих типів char, unsigned char, int і так далі. Необхідно навести приклади опису з ініціалізацією і без. Діапазони допустимих значень. Різницю між знаковими і беззнаковими типами. Операції, які можуть застосовуватися до даних типів. Правила перетворення один в інший. Додатково, можна розповісти про специфікатор місця виведення для методів Format і WriteLine.

}

МЕТОДИЧНІ МАТЕРІАЛИ ТА РЕКОМЕНДАЦІЇ

Огляд .NET. Основні поняття

ПЛАТФОРМА - це як мінімум середовище виконання програм, що визначає особливості розробки і виконання програмного коду – парадигми програмування, мови програмування, множини базових класів.

Microsoft.NET (.NET Framework) – програмна платформа, яка містить наступні основні компоненти: the common language runtime (CLR) and the .NET Framework class library (.NET FCL).

CLS (Common Language Specification) – загальна специфікація мов програмування. Це набір конструкцій і обмежень, які є керівництвом для розробників бібліотек і компіляторів в середовищі .NET Framework. Бібліотеки, побудовані відповідно до CLS, можуть бути використані з будь-якої мови програмування, що підтримує CLS. Мови, відповідні CLS (до них відносяться мови Visual C#, Visual C++ та інші), можуть інтегруватися один з одним. CLS – це основа міжмовної взаємодії в раках платформи Microsoft.NET.

CLR (Common Language Runtime) – середовище часу виконання або віртуальна машина. Забезпечує виконання збірки. Основний компонент .NET Framework. Під віртуальною вашиною розуміють абстракцію інкапсульованої (відособленою) керованої операційної системи високого рівня, яка забезпечує виконання програмного коду і підтримує вирішення наступних задач:

- управління кодом (завантаження і виконання)
- управління пам'яттю при розміщенні об'єктів
- ізоляцію пам'яті додатків
- перевірку безпеки коду
- перетворення проміжної мови в машинний код
- доступ до метаданих (розширена інформація про типи)
- обробка виключень, включаючи міжмовні виключення
- взаємодія між керованим і некерованим кодом (у тому числі і COM-об'єктами),
- підтримка сервісів для розробки (профілізація, відлагодження і т.д.).

Більш коротко, CLR – це набір служб, необхідних для виконання збірки. При цьому програмний код збірки може бути як керованим (код, при виконанні якого CLR, зокрема, активізує систему управління пам'яті), так і некерованим (“старий” програмний код).

Сама CLR складається з двох головних компонентів: ядра (mscoree.dll) і бібліотеки базових класів (mscorlib.dll). Наявність цих файлів на диску – вірна ознака того, що на комп'ютері, принаймні, була зроблена спроба установки платформи .NET.

Ядро середовища виконання реалізоване у вигляді бібліотеки mscoree.dll. При компоновці збірки в неї вбудовується спеціальна інформація,

```
}
```

яка при запуску додатку (EXE) або при завантаженні бібліотеки (звернення до DLL з некерованого модуля – виклик функції LoadLibrary для завантаження керованої збірки) приводить до завантаження і ініціалізації CLR. Після завантаження CLR в адресний простір процесу, ядро середовища виконання виконує наступні дії:

- знаходить місцезнаходження збірки
- завантажує збірку в пам'ять
- проводить аналіз вмісту збірки (виявляє класи, структури
- інтерфейси),
- проводить аналіз метаданих
- забезпечує компіляцію коду на проміжній мові (IL) в
- залежні від платформи інструкції (асемблерний код)
- виконує перевірки, пов'язані із забезпеченням безпеки
- використовуючи основний потік додатку, передає управління перетвореному в команди процесора фрагменту коду збірки.

FCL (.NET Framework Class Library) – відповідна CLS специфікації об'єктно-орієнтована бібліотека класів, інтерфейсів і системи типів (типів-значень), які включаються до складу платформи Microsoft .NET.

Ця бібліотека забезпечує доступ до функціональних можливостей системи і призначена як основа при розробці .NET додатків, компонент, елементів управління.

.NET бібліотека класів є другим компонентом CLR.

.NET FCL можуть використовувати усі .NET-додатки, незалежно від призначення, архітектури, використовуваного при розробці мови програмування. Зокрема, містить:

- вбудовані (елементарні) типи, представлені у вигляді класів (на платформі .NET все побудовано на структурах або класах),
- класи для розробки графічного призначеного для користувача інтерфейсу (Windows Forms),
- класи для розробки Web-додатків і Web-служб на основі технології ASP.NET (Web Forms),
- класи для розробки XML і Internet-протоколами (FTP, HTTP, SMTP, SOAP)
- класи для розробки додатків, що працюють з базами даних (ADO.NET)
- та багато іншого.

.NET-додаток – додаток, розроблений для виконання на платформі Microsoft.NET. Реалізується на мовах програмування, що відповідають CLS.

MSIL (Microsoft Intermediate Language, він же IL – Intermedia Language) – проміжна мова платформи Microsoft.NET. Початкові тексти програм для .NET додатків пишуться на мовах програмування, що відповідають специфікації CLS. Для таких мов програмування може бути побудований перетворювач в MSIL. Таким чином, програми на цих мовах можуть транслюватися в проміжний код на MSIL. Завдяки відповідності CLS, в

```
}
```

результаті трансляції програмного коду, написаного на різних мовах, виходить сумісний IL код. Фактично MSIL є асемблером віртуального процесора.

МЕТАДАНІ - при перетворенні програмного коду в MSIL також формується блок МЕТАДАНИХ, що містять інформацію про дані, які використовуються в програмі. Фактично це набори таблиць, що містять інформацію про типи даних, що визначені в модулі, про типи даних, на які посилається даний модуль. Раніше така інформація зберігалася окремо. Наприклад, додаток міг включати інформацію про інтерфейси, яка описувалася на Interface Definition Language (IDL). Тепер метадані є частиною керованого модуля. Зокрема, метадані використовуються для:

- збереження інформації про типи. При компіляції тепер не потрібні заголовочні і бібліотечні файли. Всю необхідну інформацію компілятор читає безпосередньо з керованих модулів
- верифікації коду в процесі виконання модуля
- управління динамічною пам'яттю (звільнення пам'яті) в процесі виконання модуля
- при розробці програми стандартними інструментальними засобами (Microsoft Visual Studio.NET) на основі метаданих забезпечується динамічна контекстна підказка (IntelliSense).

Виконуваний модуль (керований модуль) - незалежно від компілятора (і вхідної мови) результатом трансляції .NET додатку. Це стандартний мобільний виконуваний (PE – Portable Executable) файл Windows.

Керований модуль містить керований код. Керований код - це код, який виконується в середовищі CLR. Код будується на основі оголошених у початковому модулі структур і класів, що містять оголошення методів. Керованому коду повинен відповідати визначений рівень інформації (метаданих) для середовища виконання. Код C# є керованим за замовчанням. Одній з особливостей керованого коду є наявність механізмів, які дозволяють працювати з керованими даними.

Керовані дані - об'єкти, які в ході виконання коду модуля розміщуються в керованій пам'яті (у керованій купі) і знищуються прибиральником сміття CLR.

Збірка (Assembly) - базовий будівельний блок додатку в .NET Framework. Керовані модулі об'єднуються в збірки. Збірка є логічним угрупуванням одного або декількох керованих модулів або файлів ресурсів. Керовані модулі у складі збірок виконуються в середовищі часу виконання (CLR). Збірка може бути або виконуваним додатком (при цьому вона розміщується у файлі з розширенням .EXE), або бібліотечним модулем (у файлі з розширенням .DLL). При цьому нічого спільного із звичайними (старого зразку!) виконуваними додатками і бібліотечними модулями збірка не має.

Декларація збірки (Manifest) - складова частина збірки. Ще один набір таблиць метаданих, який:

```
}
```

- ідентифікує збірку у вигляді текстового імені, її версію, культуру і цифрову сигнатуру (якщо збірка розділяється серед додатків)
- визначає вхідні в склад файли (по імені і хэшу)
- вказує типи і ресурси, що існують в збірці, включаючи опис тих, які експортуються із збірки
- перераховує залежності від інших збірок
- вказує набір прав, необхідних збірці для коректної роботи.

Ця інформація використовується в період виконання для підтримки коректної роботи додатку.

Процесор не може виконувати IL код. Трансляція IL коду здійснюється JIT-компілятором (just in time – в потрібний момент), який активізується CLR по мірі необхідності та виконується процесором. При цьому результати діяльності JIT-компілятора зберігаються в оперативній пам'яті. Між фрагментом IL відтрансльованого коду та відповідним блоком пам'яті встановлюється відповідність, яка надалі дозволяє CLR передавати управління командам процесора, записаним в цьому блоці пам'яті, мінуючи повторне звернення до JIT-компілятора.

У середовищі CLR допускається спільна робота та взаємодія компонентів програмного забезпечення, реалізованих на різних мовах програмування. На основі раніше сформованого блоку метаданих CLR забезпечує ефективну взаємодію виконуваних .NET додатків. Для CLR всі складки однакові, незалежно від того, на яких мовах програмування вони були написані. Головне – це щоб вони відповідали CLS. Фактично CLR руйнує межі мов програмування (cross-language interoperability). Таким чином, завдяки CLS і CTS .NET-додатки фактично є додатками на MSIL (IL).

CLR бере на себе вирішення багатьох проблем, які традиційно знаходилися в зоні особливої уваги розробників додатків. До функцій, виконуваних CLR, відносяться:

- перевірка і динамічна (JIT) компіляція MSIL коду в команди процесора
- управління пам'яттю, процесами і потоками
- організація взаємодії процесів
- вирішення проблем безпеки (в рамках політики, що існує в системі безпеці).

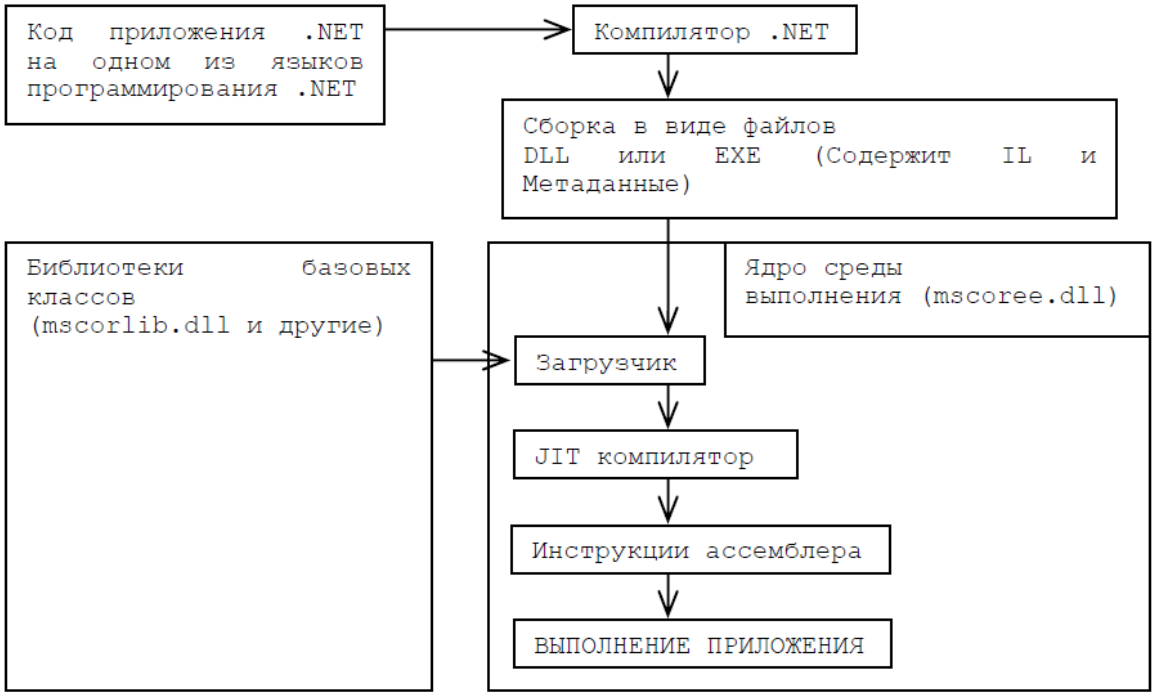
AppDomain (домен додатку) - це логічний контейнер збірок, який використовується для ізоляції додатку в рамках адресного простору процесу. Всі об'єкти, що створюються додатком, створюються в рамках певного домену додатку. Декілька доменів додатків можуть існувати в одному процесі операційної системи. CLR ізолює додатки, управляючи пам'яттю в рамках домену додатку.

Код, що виконується в CLR (CLR процес), відокремлений від інших процесів, що виконуються на комп'ютері в цей самий час. Звичайний процес запускається системою в рамках спеціально виділеного процесу адресного простору. CLR надає можливість виконання множини керованих додатків в одному процесі. Кожен керований додаток зв'язується з власним доменом

}

додатку (скорочено AppDomain). У додатку, крім основного домену, може бути створено декілька додаткових доменів.

Структура середовища виконання CLR представлена на малюнку.



Для виконання .NET-додатка достатньо розмістити збірки одного додатку в одному каталозі. Якщо при цьому збірка може бути використана в декількох додатках, то вона розміщується і реєструється за допомогою спеціальної утиліти в GAC (Global Assembly Cache - загальному кеші збірок).

CTS (Common Type System) - стандартна система типів. Підтримується всіма мовами платформи. Внаслідок того, що .NET підтримує ООП – те тут йдеться про елементарні типи, класи, структури, інтерфейси, делегатів і перерахування.

Common Type System є важливою частиною середовища виконання, визначає структуру синтаксичних конструкцій, способи оголошення, використання, і застосування загальних типів середовища виконання. У CTS зосереджена основна інформація про систему загальних зумовлених типів, про їх використання і управління (правилах перетворення значень). CTS грає важливу роль в справі інтеграції керованих додатків, розроблених з застосуванням різних мов програмування.

Простір імен – це спосіб організації системи типів в єдину групу. Існує універсальна загальномовна бібліотека базових класів. Концепція простору імен забезпечує ефективну організацію та навігацію в цій бібліотеці. Незалежно від мови програмування доступ до певних класів забезпечується за рахунок їх угрупування в рамках загальних просторів імен.

Виконання некерованих виконуваних модулів (звичайні Windows додатки), забезпечується безпосередньо системою Windows. Некеровані

```
}
```

модулі виконуються в середовищі Windows як “прості” процеси. Єдина вимога, якій повинні відповідати подібні модулі, – коректна робота в середовищі Windows. Вони повинні “правильно” працювати (не приводити до зависання системи, не допускати витоків пам'яті, не блокувати інші процеси і коректно використовувати засоби самої ОС для роботи від імені процесів). Іншими словами, відповідати найбільш загальним правилам роботи в ОС Windows. При цьому більшість проблем коректного виконання некерованого модуля (проблеми взаємодії, виділення і звільнення пам'яті) є проблемами розробників додатків.

Об'єкт – у широкому сенсі, це область пам'яті (стеку або купі), що виділяється в процесі виконання програми для запису яких-небудь значень. Характеризується типом (фіксованим набором властивостей, що визначають розмір зайнятої області, спосіб інтерпретації значення, діапазон значень, множину дій, допустимих при маніпуляціях з об'єктом) місцем розташування в пам'яті (адресою).

Прибирання сміття - механізм, який дозволяє CLR визначити, коли об'єкт стає недоступний в керованій пам'яті програми. При прибиранні сміття керована пам'ять звільняється. Для розробника додатку наявність механізму прибирання сміття означає, що він більше не повинен піклуватися про звільнення пам'яті. Проте, це може зажадати зміни в стилі програмування, наприклад, особливу увагу слід приділяти процедурі звільнення системних ресурсів. Необхідно реалізувати методи, що звільняють системні ресурси, що знаходяться під управлінням додатку.

Стек - спеціальним чином організована область пам'яті, призначена для тимчасового зберігання значень об'єктів (змінних і констант), для передачі параметрів при виклику методів, для збереження адреси повернення. Управління стеком в порівнянні з купою досить просто. Воно засноване на зміні значення відповідного регістра вершини стеку. При скороченні розміру стека об'єкти просто втрачаються.

Програма на C#

Програма – правильно побудована (що не викликає заперечень з боку C# компілятора) послідовність пропозицій, на основі якої формується збірка.

У загальному випадку, програміст створює файл, що містить оголошення класів, який подається на вхід компілятору. Результат компіляції представляється транслятором у вигляді збірки і визначається перевагами програміста. Збірка може бути двох видів:

- Portable Executable File (PE-файл з розширенням .exe), придатний до безпосереднього виконання CLR
- Dynamic Link Library File (DLL-файл з розширенням .dll), призначений для повторного використання як компонент у складі довільного додатку.

Таким чином, на основі вхідного коду транслятор будує модуль на IL та маніфест і формує збірку. Надалі, збірка або може бути виконана після JIT компіляції, або може бути використана у складі інших програм.

```
}
```

СПИСОК ЛІТЕРАТУРИ

[1] Вахромеева Л.А. Картография: Учебник для вузов. – М: Недра, 1981. - 224 с.

[2] ДСТУ ГОСТ 7.1:2006. Бібліографічний запис, бібліографічний опис. Загальні вимоги та правила складання : метод. рекомендації з впровадження / уклали: Галевич О. К., Штогрин І. М. – Львів, 2008. – 20 с.

[3] Лавринович В.Ю. Програмна реалізація машини Тьюринга/ Л.М. Соснєв, В.Ю. Лавринович// Тези XVI міжнародної науково-практичної конференції молодих учених та студентів «Політ. Сучасні проблеми науки. Інформаційно-діагностичні системи». - К.:НАУ, 2016. - С.159. http://er.nau.edu.ua:8080/handle/NAU/36796.

[4] Грінченко К.М. Перерахунок координат з системи ск-42 в wgs84 і навпаки/ К.М. Грінченко //Тези XVI міжнародної науково-практичної конф. молодих учених та студентів «ПОЛІТ. Сучасні проблеми науки. Інформаційно-діагностичні системи» - НАУ, Київ.- 2016.

[5] Лавринович В.Ю. Накладання треків з gps-координатами на растрові карти відкритих тайлових сервісів / В.Ю.Лавріновіч // Тези XVI міжнародної науково-практичної конф. молодих учених і студентів «ПОЛІТ. Сучасні проблеми науки. Інформаційно-діагностичні системи». - Київ: НАУ, 2016. - 262 с.

[6] Ковальчук Т.І. Відображення даних відеореєстратора на растрових картах / Т.І. Ковальчук, Р.О. Остапчук //XVIII міжнародна науково-практична конференція молодих учених і студентів «політ-2018». – Тезиси докладів. К.:НАУ, 2018.

[7] Культин Н. Б. Microsoft Visual C# в задачах и примерах. – СПб.: БХВ-Петербург, 2009. – 320 с.

[8] Дейт К. Дж. Введение в системы баз данных . An Introduction to Database System, 7th Edition. . — М.: Вильямс, 2001. — С. 1072. — ISBN 5-8459-0138-3.

[9] Пискунов А.Г. Библиотека обработки аргументов командной строки. / А.Г.Пискунов, Д.И.Гись // LXXIV-а наукова конференція професорсько-викладацького складу, аспірантів, студентів та співробітників відокремлених структурних підрозділів університету. – К.: НТУ, 2018. – с.421. - Режим доступа: http://agp1.adr.com.ua/args.html.

[10] Пискунов А.Г. Doxygen и Graphviz: документирование проектов на C# [Электрон.ресурс]/ А.Г. Пискунов, А.С.Горбань // Журн. Алгоритм.– №4(10), – 2006. - Режим доступа: http://www.realcoding.net/dn/docs/dgIntro.pdf.

[11] Пискунов А.Г. Graphviz и Sed: построение схем иерархии наследования [Электрон.ресурс]/ А.Г. Пискунов, С.М. Петренко // Журн. Алгоритм. – №3(9). – 2006. - Режим доступа: http://www.softcraft.ru/design/graphvizsed/mkhrrchy.pdf.

}

[12] Пискунов А.Г. Об отличиях между понятиями типа и класса / Вестник Киевского университета. Компьютерные науки. – 2015. – № 3. Интернет ресурс: http://irbis-nbuv.gov.ua/cgi-in/irbis_nbuv/cgiirbis_64.exe? I21DBN=LINK P21DBN=UJRN Z21ID= S21REF=10 S21CNR=20 S21STN=1 S21FMT=ASP_meta_C21COM=S2_S21P03=FILA=2_S21STR=VKNU_fiz_mat_ 2015_3_22.

[13] Пискунов А.Г. Пояснительная записка. Эллипсоиды и геометрические вычисления / А.Г.Пискунов, К.М.Гринченко, И.А.Юрчук. // Кафедра Прикладной Математики. - Киев: ИИДС - 2017 - 248 с.

[14] Пискунов А.Г. Опыт эксплуатации одной системы журналирования приложений / А.Г.Пискунов, Р.В.Скаковский. // Проблеми інфокомунікацій: Матеріали другої всеукраїнської науково-технічної конференці. – Полтава: ПолтНТУ; Київ: НТУ; Харків: НТУ «ХПІ»; Полтава: ВКСС ВІТІ, 2018. – С.74. - Режим доступа: https://drive.google.com/open? id=10eE0nSzhkZlz8xR5qOqcSg4jQ2kd5Ecj.

[15] Пискунов А.Г. Проектирование и декомпозиция двунаправленных потоков данных интерактивных систем [Электрон.ресурс] / 2009. - Режим доступа: http://er.nau.edu.ua:8080/bitstream/NAU/39629/1/ dataFlow.pdf.

[16] Пискунов А.Г. Система ограничения доступа Вежа. Пояснительная записка [Электрон.ресурс] / 2000. - Режим доступа: http://agp1.adr.com.ua/ software/mlc.pdf.

[17] Пискунов А.Г. SQLite и введение в разработку клиент - серверных приложений на языке C# [Электрон.ресурс] /2019. - Режим доступа: http:// agp1.adr.com.ua/docs/db.Labs.pdf.

[18] Пискунов А.Г. Разработка консольных и оконных приложений на языке C# [Электрон.ресурс]/ 2019. - Режим доступа: http://agp1.adr.com.ua/ docs/smp.Labs.pdf.

[19] Пискунов А.Г. Компиляторы C++ и разработка консольных приложений (Часть 1) [Электрон.ресурс]/ 2020. - Режим доступа: http://agp1.adr.com.ua/docs/amp.Lctrs.pdf.

[20] Пискунов А.Г. Разработка консольных приложений с элементами C++ (часть2) [Электрон.ресурс]/ 2018. - Режим доступа: http://gp1.adr.com.ua/ ocs/ppz.Lctrs.pdf.

[21] Пискунов А.Г. Журналирование многосредовых приложений. Пояснительная записка [Электрон.ресурс]/ А.Г.Пискунов, Р.В.Скаковский. - Режим доступа: http://agp1.adr.com.ua/Logger.html.

[22] Math Parser. Math Parser.NET. [Электрон.ресурс]. - Режим доступа: https:// www.codeproject.com/Articles/ 274093/Math-Parser-NET.

[23] Документація фірми Microsoft. Руководство по программированию на C# [Электрон.ресурс]. - Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/.

}

[24] Кост С. Некоммерческий картографический проект OSM [Электрон. ресурс]/ С.Кост. - Режим доступа: http://www.openstreetmap.org.

[25] Сусідик Н.С. Автоматизований збір висот точок місцевості/ Н.С. Сусідик // XVIII міжнародна науково-практична конференція молодих учених і студентів «політ-2018». – Тезиси докладів. К.:НАУ, 2018.

[26] Троелсен Э. C# и платформа. NET. Библиотека программиста / Пер. с англ. . – СПб.: Питер, 2004. – 796 с.

[27] Шилдт Г. Полный справочник по C# / Пер. с англ. . – М.: Издательский дом ”Вильямс”, 2004. – 752 с.

[28] Рихтер Дж. Программирование на платформе Microsoft .NET Framework / Пер. с англ. . – 2-е изд., испр. – М.: Издательско – торговый дом “Русская редакция”, 2003. – 512 с.

[29] [Электронный ресурс]. – Режим доступа: agp1.adr.com.ua/tools/ resXGen.zip

[30] Документація фірми MicroSoft. Quickstart: Docker in Visual Studio. - Режим доступа: https://docs.microsoft.com/en-us/visualstudio/containers/ container-tools?view=vs-2019&viewFallbackFrom=vs-2019.md

[31 Документація фірми Microsoft. Host ASP.NET Core in Docker containers. - Режим доступа: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/? view=aspnetcore-3.1

[32 Драйвер Npgsql [Электронный ресурс]. – Режим доступа: http://pgfoundry.org/frs/?group_id=1000140.

[33 Работа с MS Word в C# [Электронный ресурс]. – Режим доступа: http://wladm.narod.ru/C_Sharp/comword.html.

}