ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

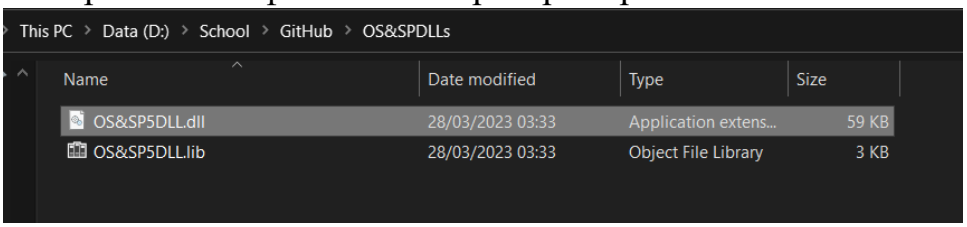
Кафедра дискретного аналізу

Операційні системи та системне програмування
Лабораторна робота №7

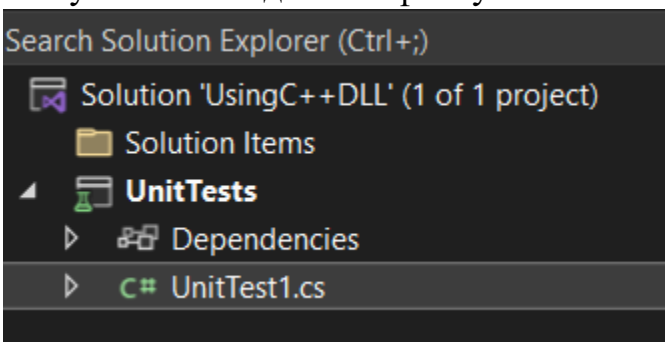
Виконав
Студент групи ПМІ-43
Заречанський Олексій
Викладач
Доц. Черняхівський Володимир

2023

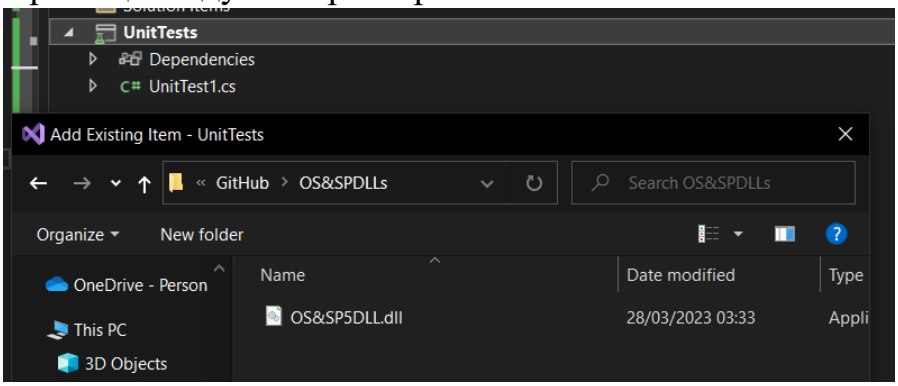
1. Використаю dll файл з 5 лабораторної роботи.



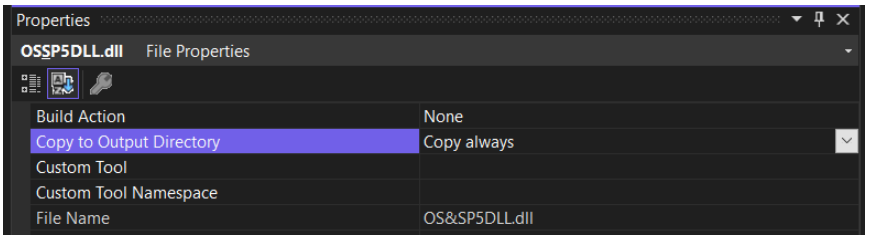
Створюю нове рішення в Visual Studio та проект з юніт тестами для тестування методів з dll файлу.



2. Для того щоб dll була знайдена можна помістити її в папку з системними dll.
   Наприклад netapi32.dll, яка наприклад містить методи для отримання часу з ремоут серверу, знаходиться в папках C:\Windows\SysWOW64 та C:\Windows\System32. Якщо помістити нашу dll в одну з цих папок то вона буде знаходитись проектом.
   Інше рішення спробувати задати локацію dll в змінну середовища Path. Проте це не дуже хороші рішення.



У своєму випадку я просто додав цей файл до проекту в Visual Studio.

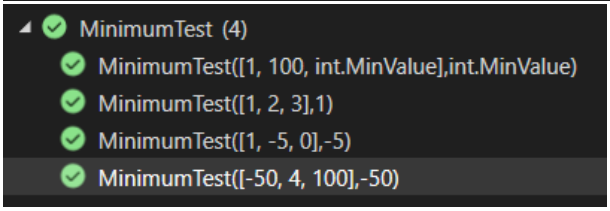


Після чого у властивостях файла вкажу копіювати його в папку з результатом білду програми.

```
[DllImport("OS&SP5DLL.dll")]
0 references
private static extern int Minimum(int[] intArr, int arraySizeInBytes);

[DllImport("OS&SP5DLL.dll")]
0 references
private static extern bool Contains(string text, int textSize, string toFind, int searchedStringSize, bool caseSensitive);

[DllImport("OS&SP5DLL.dll")]
0 references
private static extern double Average(int[] intArr, int arraySizeInBytes);

[DllImport("OS&SP5DLL.dll")]
1 reference
private static extern int Count(string text, int textSize, char searched, bool caseSensitive);
```

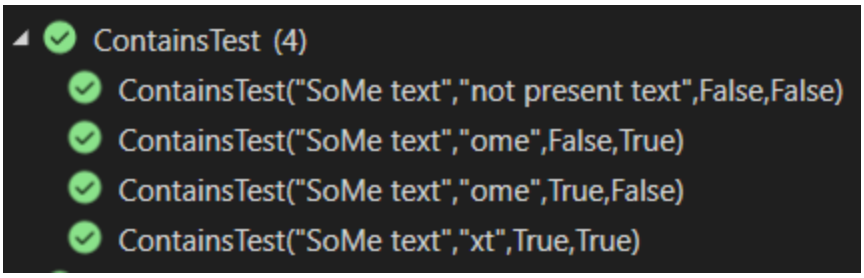
Після чого оголошую кожну функцію окремо для неявного зв’язування.
Далі напишу юніт тести на тестування кожного з методів.
Тестування функції Minimum.

```
[TestCase(new[] { 1, 2, 3}, 1)]
[TestCase(new[] { 1, -5, 0 }, -5)]
[TestCase(new[] { 1, 100, int.MinValue }, int.MinValue)]
[TestCase(new[] { -50, 4, 100 }, -50)]

| 0 references
public void MinimumTest(int[] intArr, int shouldBeResult)
{
    int min = Minimum(intArr, intArr.Length * sizeof(int));

    Assert.That(min, Is.EqualTo(shouldBeResult));
}
```



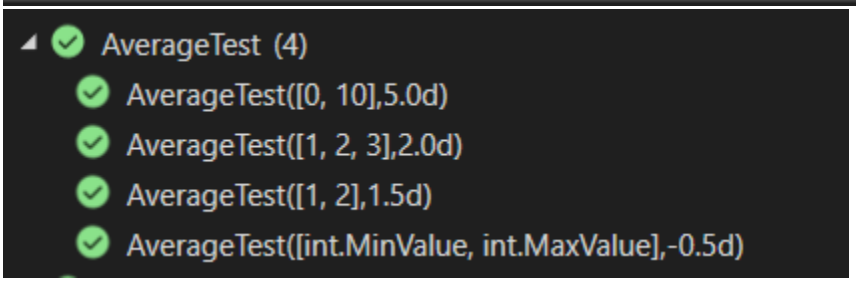
Тестування функції Contains.

```
[TestCase("SoMe text","ome", true, false)]
[TestCase("SoMe text", "ome", false, true)]
[TestCase("SoMe text", "not present text", false, false)]
[TestCase("SoMe text", "xt", true, true)]
| 0 references
public void ContainsTest(string text, string textToFind, bool caseSensitivity, bool containResult)
{
    bool isContained = Contains(text, text.Length + 1, textToFind, textToFind.Length + 1, caseSensitivity);

    Assert.That(isContained, Is.EqualTo(containResult));
}
```



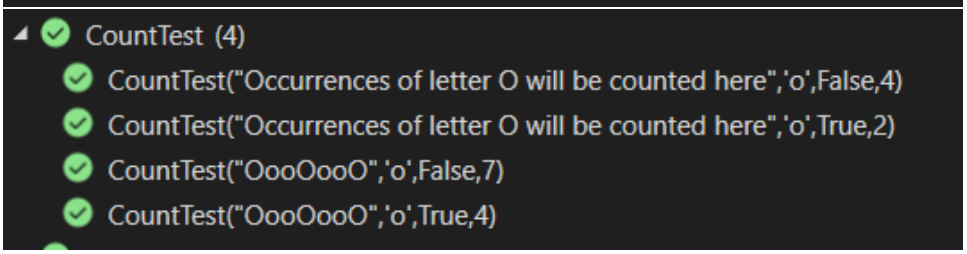
Тестування функції Average.

```
[TestCase(new[] { 1, 2, 3 }, 2.0)]
[TestCase(new[] { 0, 10 }, 5.0)]
[TestCase(new[] { int.MinValue, int.MaxValue }, -0.5)]
[TestCase(new[] { 1, 2 }, 1.5)]
0 references
public void AverageTest(int[] intArr, double shouldBeResult)
{
    double avg = Average(intArr, intArr.Length * sizeof(int));

    Assert.That(avg, Is.EqualTo(shouldBeResult));
}
```

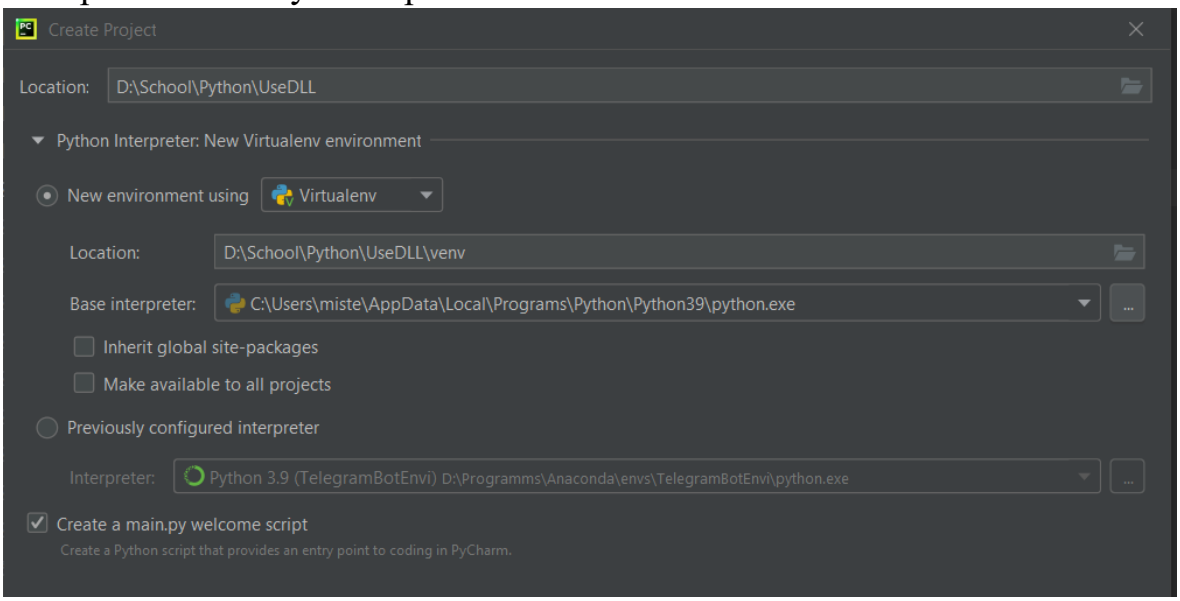


Тестування функції Count.

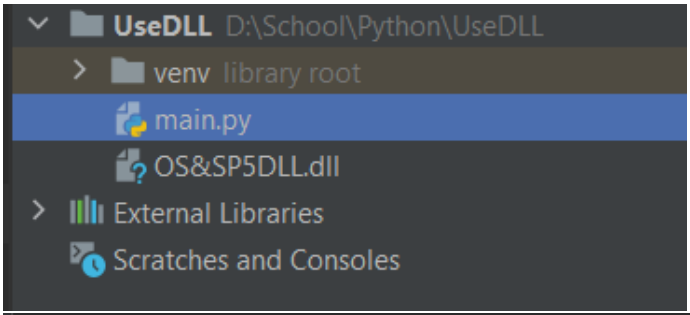
```
[TestCase("Occurrences of letter O will be counted here", 'o', true, 2)]
[TestCase("Occurrences of letter O will be counted here", 'o', false, 4)]
[TestCase("OooOooO", 'o', true, 4)]
[TestCase("OooOooO", 'o', false, 7)]
0 references
public void CountTest(string text, char searched, bool caseSensitive, int counted)
{
    int occurences = Count(text, text.Length + 1, searched, caseSensitive);

    Assert.That(occurences, Is.EqualTo(counted));
}
```



3. Створюю новий Python проект



З версії Python 3.8 більше не шукаються dll шляхом додавання їх у змінну середовища Path, є 2 способи, задавати абсолютний шлях до dll або задавати шлях до папок які містять dll за допомогою os.add_dll_directory.
Я обрав перший спосіб, помістив dll в одну папку з .py файлом, та задав шлях відносно нього. Після чого завантажив бібліотеку.




```
dirName = os.path.dirname(__file__)
dllPath = os.path.join(dirName, "OS&SP5DLL.dll")
hDLL = ctypes.WinDLL(dllPath)
```

Після чого я створив вказівники на функції, задав їм типи повернення

та для двох з них задав типи вхідних параметрів. Я не зробив цього для 2 інших функцій, оскільки параметрами для них є вектори, розміри яких потрібно задавати власне перед передачею параметрів.

```
Minimum = hDLL.Minimum
Minimum.restype = ctypes.c_int
Contains = hDLL.Contains
Contains.restype = ctypes.c_bool
Contains.argtypes = [ctypes.c_char_p, ctypes.c_int, ctypes.c_char_p, ctypes.c_int, ctypes.c_bool]
Average = hDLL.Average
Average.restype = ctypes.c_double
Count = hDLL.Count
Count.restype = ctypes.c_int
Count.argtypes = [ctypes.c_char_p, ctypes.c_int, ctypes.c_char, ctypes.c_bool]
```

Обгортка для методу Minimum, яка встановлює довжину вхідного вектору для методу, та будує цей вектор з звичайного списку Python, тепер для виклику Minimum потрібно лише буде викликати його обгортку передавши список цілих значень, мінімальне з яких бажаємо отримати.

```
def minimum(values):
    length = len(values)
    Minimum.argtypes = [ctypes.c_int*length, ctypes.c_int]
    arr = (ctypes.c_int * length)()

    for i in range(length):
        arr[i] = values[i]

    return Minimum(arr, length * 4)
```

Обгортка для методу Contains, перед використанням потрібно перетворити літери Python які є у UTF-8 в Ansi, який використовується в C. Після цього можемо створити char* та передати його в метод Contains. Викликаємо обгортку лише передаючи текст в Python та True або False як case_sensitivity.

```
def contains(text, to_find, case_sensitivity):

    return Contains(ctypes.c_char_p(text.encode('ansi')),
                    len(text) + 1,
                    ctypes.c_char_p(to_find.encode('ansi')),
                    len(to_find) + 1,
                    case_sensitivity)
```

Обгортка для методу Average, яка встановлює довжину вхідного

вектору для методу, та будує цей вектор з звичайного списку Python, тепер для виклику Average потрібно лише буде викликати його обгортку передавши список цілих значень, середнє з яких бажаємо отримати.

```
def average(values):
    length = len(values)
    Average.argtypes = [ctypes.c_int*length, ctypes.c_int]
    arr = (ctypes.c_int * length)()

    for i in range(length):
        arr[i] = values[i]

    return Average(arr, length * 4)
```

Обгортка для методу Count, перед використанням потрібно перетворити літери Python які є у UTF-8 в Ansi, який використовується в C. Після цього можемо створити char* для тексту та char для символу який рахуємо та передати їх в метод Count. Викликаємо обгортку лише передаючи текст в Python та True або False як case_sensitivity.

```
def count(text, char, case_sensitivity):
    return Count(ctypes.c_char_p(text.encode('ansi')),
                 len(text) + 1,
                 ctypes.c_char(char.encode('ansi')),
                 case_sensitivity)
```

4. Для тестування я написав той самий код що був в 5 лабораторній для тестування функцій в C++, але вже на пайтоні.
   Тестування Minimum (викликаємо через обгортку)

```
if __name__ == '__main__':
    minValue = minimum([3, 1, -5, 6, 0])
    if minValue != -5:
        print(f'Error in Minimum function, result has to be -5, not {minValue}')
    else:
        print('Minimum function works correctly')
```

   Тестування Contains(викликаємо через обгортку)

```
textToSearch = 'SoMe text'
textToFind = 'ome'
resultContains = contains(textToSearch, textToFind, False)
if resultContains is True:
    print('Contains function works correctly with no regard for case')
else:
    print(f'Error in Contains function when case is not taken into account, ' +
          f'result has to be false, not {resultContains}')

resultContains = contains(textToSearch, textToFind, True)
if resultContains is False:
    print('Contains function works correctly with case sensitive values')
else:
    print(f'Error in Contains function when case is taken into account, ' +
          f'result has to be false, not {resultContains}')
```

Тестування Average(викликаємо через обгортку)

```
averageNum = average([8, 5, 4, 2])
if averageNum == 4.75:
    print('Average function works correctly')
else:
    print(f'Average function works incorrectly, result has to be 4.75, not {averageNum}')
```

Тестування Count(викликаємо через обгортку)

```
textToFilter = 'Occurrences of letter O will be counted here'
letter = 'o'
occurrences = count(textToFilter, letter, True)
if occurrences == 2:
    print('Count function works correctly with case sensitive values')
else:
    print(f'Error in Count function when case is taken into account, result has to be 2, not {occurrences}')

occurrences = count(textToFilter, letter, False)
if occurrences == 4:
    print('Count function works correctly with case insensitive values')
else:
    print(f'Error in Count function when case is not taken into account, result has to be 4, not {occurrences}')
```

Результат виконання - все працює добре.

```
main
D:\School\Python\UseDLL\venv\Scripts\python.exe D:/School/Python/UseDLL/main.py
Minimum function works correctly
Contains function works correctly with no regard for case
Contains function works correctly with case sensitive values
Average function works correctly
Count function works correctly with case sensitive values
Count function works correctly with case insensitive values

Process finished with exit code 0
```

5. Не секрет що C++ це мова набагато нижчого рівня, яка дозволяє працювати з об’єктами точніше, ніж це дозволяють інші мови, проте має набагато менший функціонал ніж наприклад Python з його різномаїттям бібліотек. Також код на C++ виконується в рази швидше (один і той самий код виконується в десятки разів швидше на C++ ніж на умовному Python). Тому у мовах інтерпретованого типу можна використовувати DLL, це доцільно в ситуаціях коли нам потрібна максимальна швидкодія або у нас ми виконуємо операції які вимагають дуже багато обчислень. Це допоможе програмувати на більш зручних мовах програмування та використовувати їх поширений функціонал, але за потреби також використати швидкодію DLL для важких операцій.