

## Лабораторна робота №1

**Тема:** Інтегроване середовище розробки Visual Studio. Створення і компіляція консольного проекту за допомогою .net cli.

**Мета:** Створення GitHub репозиторію і завантаження коду проекту в нього. Підключення CI системи Travis до github репозиторію.

**Завдання:** Здобути навички створення, відлагодження та тестування простих проектів мовою C#.  
(max: 5 балів)

### Теоретичні відомості:

Для запису програми в C# використовується кодування символів Unicode. Кодуванням, або *кодуючою таблицею (character set)*, називається відповідність між символами і числами, що кодують їх. Кодування Unicode дозволяє представити символи всіх існуючих алфавітів одночасно. Кожному символу відповідає свій унікальний код.

Алфавіт C# включає:

- *букви* (латинські і національних алфавітів) і символ підкреслення (\_), який вживається разом з буквами;
- *цифри*;
- *спеціальні символи*, наприклад +, \*, { і &;
- *пробільні символи* (пробіл і символи табуляції);
- *символи переведення рядка*.

З символів складаються більші будівельні блоки: лексеми, директиви препроцесора і коментарі.

*Лексема (token)* — це мінімальна одиниця мови, що має самостійний сенс. Існують такі види лексем:

- *імена (ідентифікатори)*;
- *ключові слова*;
- *знаки операцій*;
- *роздільники*;
- *літерали (константи)*.

#### **Ідентифікатори**

Імена (або ідентифікатори) використовуються для того, щоб звертатися до програмних об'єктів і розрізнити їх, тобто ідентифікувати. В ідентифікаторі дозволяється використовувати букви, цифри і символ підкреслення. Прописні і рядкові букви **розрізняються**, наприклад, hacker, Hacker і hAcKeR - три різні імена.

Першим символом ідентифікатора може бути буква або знак підкреслення, але не цифра. Довжина ідентифікатора не обмежена. Пропуски усередині імен не допускаються.

#### **Ключові слова**

*Ключові слова* — це зарезервовані ідентифікатори, які мають спеціальне значення для компілятора. Їх можна використовувати лише в тому значенні, в якому вони визначені.

**Таблиця 1. Ключові слова C#**

abstract	As	base	bool	break
byte	Case	catch	char	checked
class	Const	continue	decimal	default
delegate	Do	double	else	enum
event	Explicit	extern	false	finally
fixed	Float	for	foreach	goto
if	Implicit	in	int	interface
internal	Is	lock	long	namespace
new	Null	object	operator	out
override	Params	private	protected	public

readonly	Ref	return	sbyte	sealed
short	Sizeof	stackalloc	static	string
struct	Switch	this	throw	true
try	Typeof	uint	ulong	unchecked
unsafe	Ushort	using	virtual	void
volatile	While			

### Знаки операцій і роздільники

*Знак операції* — це один або більше символів, що визначають дію над операндами. У середині знаку операції пробіли не допускаються. Символи, які складають знак операцій, можуть бути спеціальними, наприклад +, &&, | i <, і буквеними, такими як as або new.

Операції діляться на унарні, бінарні і тернарні за кількістю операндів, що беруть участь в них (один, два і три операнди відповідно). Один і той самий знак може інтерпретуватися по-різному залежно від контексту.

Роздільники використовуються для розділення або, навпаки, групування елементів. Приклади роздільників: дужки, крапка, кома. Нижче перераховані всі знаки операцій і роздільники, що використовуються в C#:

```
{ } [ ] ( ) . , ; + - * / % & | ^ ! ~ =
< > ? ++ -- && || << >> == != <= >= += -= *= /= %=
&= |= ^= <=< >=> ->
```

### Літерали (константи)

*Літералами, або константами*, називають незмінні величини. У C# є логічні, цілі, дійсні, символьні і строкові константи, а також константа null. Компілятор, виділивши константу як лексему, відносить її до одного з типів даних за її зовнішнім виглядом. Програміст може задати тип константи і самостійно.

*Логічні літерали: true і false.* Вони широко використовуються як ознаки наявності або відсутності чого-небудь.

*Цілі літерали* можуть бути представлені або в десятковій, або в шістнадцятковій системі числення, а дійсні — лише в десятковій системі, але в двох формах: з фіксованою крапкою і з порядком.

Константа null є значенням, що задається за замовчанням для величин **посилкових** типів, яких ми розглянемо далі в цій лекції.

### Коментарі

*Коментарі* призначені для запису пояснень до програми і формування документації. Компілятор коментарі ігнорує. У середині коментаря можна використовувати будь-які символи. У C# є два види коментарів: однорядкові і багаторядкові.

*Однорядковий коментар* починається з двох символів косої риски (//) і закінчується символом переходу на новий рядок, багаторядковий розміщується між символами-дужками /\* і \*/ і може займати частину рядка, цілий рядок або декілька рядків. Коментарі не вкладаються один в один.

Крім того, в мові є ще один вид коментарів, які починаються з трьох символів косої риски, що йдуть підряд (///). Вони призначені для формування документації до програми у форматі XML. Компілятор витягує ці коментарі з програми, перевіряє їх відповідність правилам і записує їх в окремий файл.

### Символьні Escape-послідовності

Escape-послідовності використовуються для створення додаткових ефектів (дзвінок), простого форматування інформації, що виводиться, і кодування символів при виведенні і порівнянні (у виразах порівняння). В таблиці наведені основні Escape-послідовності.

\a	Попередження (дзвінок)
\b	Повернення на одну позицію
\f	Перехід на нову сторінку
\n	Перехід на новий рядок
\r	Повернення каретки

\t	Горизонтальна табуляція
\'	Одинарна кавичка
\''	Подвійна кавичка

Дані, з якими працює програма, зберігаються в оперативній пам'яті. Природно, що компілятору необхідно точно знати, скільки місця вони займають, як саме закодовані і які дії з ними можна виконувати. Все це задається при описі даних за допомогою типу.

Тип даних однозначно визначає:

- внутрішнє представлення даних, а отже і множину їх можливих значень;
- допустимі дії над даними (операції і функції).

Наприклад, цілі і дійсні числа, навіть якщо вони займають однаковий об'єм пам'яті, мають абсолютно різні діапазони можливих значень.

Кожний вираз в програмі має певний тип. Компілятор використовує інформацію про тип при перевірці допустимості описаних в програмі дій.

Пам'ять, в якій зберігаються дані під час виконання програми, ділиться на дві області: стек (stack) і динамічна область, або хіп (heap), частіше називається купою. Стек використовується для зберігання величин, пам'ять під які виділяє компілятор, а в динамічній області пам'ять резервується і звільняється під час виконання програми за допомогою спеціальних команд.

Всі типи можна розділити на *прості* (які не мають внутрішньої структури) і *структуровані* (складаються з елементів інших типів). За своїм "творцем" типи можна розділити на *вбудовані* (стандартні) і типи, які *визначаються програмістом*. За способом зберігання значень типи діляться на *типи-значення* (статичні), і *посилкові* (динамічні). Розглянемо в першу чергу вбудовані типи C#.

#### Вбудовані типи

Вбудовані типи не вимагають попереднього визначення. Для кожного типу існує ключове слово, яке використовується при описі змінних і констант. Вбудовані типи C# наведені в таблиці 2. Вони однозначно відповідають стандартним класам бібліотеки .NET, яка визначена в просторі імен System. Як видно з таблиці, існують декілька варіантів представлення цілих і дійсних величин. Програміст вибирає тип кожної величини, яка використовується в програмі, з урахуванням необхідного йому діапазону і точності представлення даних.

Таблиця 2. Вбудовані типи C#

Назва	Ключове слово	Тип .NET	Діапазон значень	Опис	Розмір, бітів
<b>Логічний тип</b>	Bool	Boolean	true, false		
<b>Цілі типи</b>	Sbyte	SByte	від -128 до 127	Зі знаком	8
	Byte	Byte	від 0 до 255	Без знака	8
	Short	Int16	від -32768 до 32767	Зі знаком	16
	Ushort	UInt16	від 0 до 65535	Без знака	16
	Int	Int32	від $-2 \times 10^9$ до $2 \times 10^9$	Зі знаком	32
	UInt	UInt32	від 0 до $4 \times 10^9$	Без знака	32
	Long	Int64	від $-9 \times 10^{18}$ до $9 \times 10^{18}$	Зі знаком	64
	Ulong	UInt64	від 0 до $18 \times 10^{18}$	Без знака	64
<b>Символьний тип</b>	Char	Char	від U+0000 до U+ffff	Unicode-символ	16
<b>Дійсні числа</b>	Float	Single	від $1.5 \times 10^{-45}$ до $3.4 \times 10^{38}$	7 цифр	32
	Double	Double	від $5.0 \times 10^{-324}$ до $1.7 \times 10^{308}$	15–16 цифр	64
<b>Фінансовий тип</b>	decimal	Decimal	від $1.0 \times 10^{-28}$ до $7.9 \times 10^{28}$	28–29 цифр	128
<b>Рядковий тип</b>	String	String	Довжина обмежена об'ємом доступної пам'яті	Рядок Unicode-символів	із
<b>Тип <i>object</i></b>	Object	Object	Можна зберігати все, що завгодно	Загальний предок	

### Примітка

Дійсні типи і фінансовий тип є знаковими. Для них в стовпчику "діапазон значень" наведені абсолютні величини допустимих значень.

Логічний, або булевий, тип містить всього два значення: true (істина) і false (не істина).

Всі цілі і дійсні типи разом з символьним і фінансовим можна назвати *арифметичними типами*.

Внутрішнє представлення величини цілого типу — ціле число в двійковому коді. У знакових типах старший біт числа інтерпретується як знак (0 - позитивне число, 1 - негативне).

Дійсні типи, або типи даних, з плаваючою крапкою зберігаються в пам'яті комп'ютера інакше, ніж цілі. Внутрішнє представлення дійсного числа складається з двох частин - мантиси і порядку, причому кожна частина має знак. Довжина мантиси визначає точність числа, а довжина порядку - його діапазон. Всі дійсні типи можуть представляти як додатні, так і від'ємні числа. Найчастіше в програмах використовується тип **double**, оскільки його діапазон і точність покривають більшість потреб.

Тип *decimal* призначений для грошових обчислень, в яких критичні помилки округлення. Величини типа *decimal* дозволяють зберігати 28–29 десяткових розрядів. Тип *decimal* не відноситься до дійсних типів, у них різне внутрішнє представлення. Величини грошового типа навіть не можна використовувати в одному виразі з дійсними без явного перетворення типа.

Будь-який вбудований тип C# відповідає стандартному класу бібліотеки .NET, визначеному в просторі імен System.

### Прикладами типів в C# є:

#### Типи-Значення (*value types*):

- прості (базові): `int i; float x;`
- перерахування: `enum State { Off, On }`
- структури: `struct Point {int x,y;}`

#### Посилкові типи (*reference types*):

- кореневі: `object`
- рядкові: `string`
- класи: `class Foo: Bar, IFoo {...}`
- інтерфейси: `interface IFoo: IBar {...}`
- масиви: `string[] = new string[10];`
- делегати: `delegate void Empty();`

Зверніть увагу, що типи `string` і масиви є посилковими.

Посилкові типи *object* та *string* є наперед визначеними (вбудованими). Вони належать до простору імен System, отже їх методи доступні для всіх об'єктів C# (вони позначаються `System.Object` и `System.String`.)

**Змінна** — це іменована область пам'яті, призначена для зберігання даних певного типу. Під час виконання програми значення змінної можна змінювати.

Мова C# є строго типізованою мовою з чітким контролем типів даних. Усі змінні повинні бути визначені до їх першого використання.

При описі для кожної змінної задаються її ім'я і тип.

Визначення змінних *простих типів* (вбудованих типів-значень) має наступний загальний синтаксис:

*Тип\_змінної* *ім'я\_змінної* [=значення];

Ім'я змінної служить для звернення до області пам'яті, в якій зберігається значення змінної. Ім'я дає програміст. Тип змінної вибирається, виходячи з діапазону і необхідної точності представлення даних. При оголошенні можна привласнити змінній деяке початкове значення, тобто ініціалізувати її.

### Наприклад:

```
int a, b = 1;
float x = 0.1, y = 0.1f;
int x;
x=100;
```

```
long w,z=100;
long q=100*z;
```

Рекомендується завжди ініціалізувати змінні при описі. При ініціалізації можна використовувати не лише константу, але і вираз— головне, щоб на момент опису воно було обчислюваним, наприклад:

```
int b = 1, a = 100;
int x = b * a + 25;
```

Програма на C# складається з класів, усередині яких описують методи і дані. Змінні, описані безпосередньо усередині класу, називаються полями класу. Їм автоматично призначається так зване "значення за замовчанням" — як правило, це 0 відповідного типу. Змінні, описані усередині методу класу, називаються локальними змінними. Їх ініціалізація покладається на програміста.

**Область дії змінної в C# - блок коду ({}).** Змінна створюється при вході в область видимості і знищується при виході з неї.

Область дії змінної, тобто область програми, де можна використовувати змінну, починається в точці її опису і триває до кінця блоку, усередині якого вона описана.

**Блок** — це код, взятий у фігурні дужки. Основне призначення блоку — угруповання операторів. У C# будь-яка змінна описана усередині якого-небудь блоку: класу, методу або блоку усередині методу. Ім'я змінної має бути унікальним в області її дії. Область дії поширюється на вкладені в метод блоки.

#### Увага

Змінні створюються при вході в блок і знищуються при виході. Це означає, що після виходу з блоку значення змінної не зберігається. При повторному вході в цей же блок змінна створюється заново.

#### Іменовані константи

Можна заборонити змінювати значення змінної, задавши при її описі ключове слово `const`, наприклад:

```
const int b = 1;
const float x = 0.1, v = 0.1f; // const поширюється на обидві змінні
```



Такі величини називають іменованими константами, або просто константами. Вони застосовуються для того, щоб замість значень констант можна було використовувати в програмі їх імена. Це робить програму зрозумілішою і полегшує внесення до неї змін. Іменовані константи повинні обов'язково ініціалізуватися при описі, наприклад:

```
const int b = 1, a = 100;
const int x = b * a + 25;
```

**Вираз** — це правило обчислення значення. У виразі беруть участь операнди, об'єднані знаками операцій. Операндами простого виразу можуть бути константи, змінні і виклики функцій.

Наприклад, `a+2` — це вираз, в якому `+` є знаком операції, а `a` і `2` — операндами. Пропуски усередині знаку операції, що складається з декількох символів, не допускаються. Операції C# наведені в таблиці 3.

Таблиця 3. Операції C#		
Категорія	Знак операції	Назва
Первинні		Доступ до елементу
	<code>x()</code>	Виклик методу або делегата
	<code>x[]</code>	Доступ до елементу
	<code>x++</code>	Постфіксний інкремент
	<code>x--</code>	Постфіксний декремент
	<code>new</code>	Виділення пам'яті

	typeof	Отримання типу
	checked	Код, що перевіряється
	unchecked	Код, що не перевіряється
Унарні	+	Унарний плюс
	-	Унарний мінус (арифметичне заперечення)
	!	Логічне заперечення
	?	Порозрядне заперечення
	++x	Префіксний інкремент
	--x	Префіксний декремент
	(тип)x	Перетворення типу
Мультиплікативні	*	Множення
	/	Ділення
	%	Залишок від ділення
Додавання/віднімання	+	Додавання
	-	Віднімання
Зсув	<<	Зсув вліво
	>>	Зсув вправо
Відношення і перевірка типу	<	Менше
	>	Більше
	<=	Менше або рівно
	>=	Більше або рівно
	is	Перевірка належності до типу
	as	Приведення типу
Перевірки на рівність	==	Рівно
	!=	Не рівно
Порозрядні логічні	&	Порозрядна кон'юнкція (І)
		Порозрядне АБО
		Порозрядна диз'юнкція (АБО)
Умовні логічні	&&	Логічне І
		Логічне АБО
Умовна	? :	Умовна операція
Присвоєння	=	Присвоєння
	*=	Множення з присвоєнням
	/=	Ділення з присвоєнням
	%=	Залишок відділення з присвоєнням
	+=	Складання з присвоєнням
	-=	Віднімання з присвоєнням
	<<=	Зсув вліво з присвоєнням
	>>=	Зсув вправо з присвоєнням
	&=	Порозрядне І з присвоєнням
	 =	порозрядне виключаюче АБО з присвоєнням
	=	Порозрядне АБО з присвоєнням

При обчисленні виразів може виникнути необхідність в перетворенні типів. Якщо операнди, що входять у вираз, одного типу, і операція для цього типу визначена, то результат виразу матиме той самий тип.

Якщо операнди різного типу і (або) операція для цього типу не визначена, перед обчисленнями автоматично виконується перетворення типу по правилах, що забезпечують приведення коротших типів до довших для збереження значущості і точності. Автоматичне (неявне) перетворення можливе не завжди, а лише якщо при цьому не може статися втрата значущості.

Якщо неявного перетворення з одного типу в інший не існує, програміст може задати явне перетворення типу за допомогою операції (тип) x.

Розрізняються:

*Розширююче перетворення* – значення одного типу приводиться до значення іншого типу, яке має такий самий або більший розмір. Наприклад, значення, представлене у вигляді 32-розрядного цілого числа із знаком, може бути перетворено в 64-розрядне ціле число із знаком. Таке перетворення вважається безпечним, оскільки початкова інформація при такому перетворенні не спотворюється.

#### Можливі варіанти перетворення типів

Byte	UInt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
SByte	Int16, Int32, Int64, Single, Double, Decimal
Int16	Int32, Int64, Single, Double, Decimal
UInt16	UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Char	UInt16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Int32	Int64, Double, Decimal
UInt32	Int64, Double, Decimal
Int64	Decimal
UInt64	Decimal
Single	Double

Всі арифметичні класи, у тому числі клас Int, мають перегружений статичний метод Parse, в якого першим обов'язковим параметром є рядок, що задає значення відповідного арифметичного типу в локалізованій формі. Форматом рядка і стилем її представлення можна управляти за допомогою інших параметрів методу Parse.

Ось приклад виклику цього методу для класів Int і Double.

#### Приклад 1.

// Перетворення типу з використанням методу Parse

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Parsing
{
    class Program
    {
        static void Main(string[] args)
        {
            //method Parse
            Console.WriteLine("Введіть ціле");
            string strdata = Console.ReadLine();
            int intdata = int.Parse(strdata);
            Console.WriteLine("Введіть число з дробовою частиною
і порядком");
            strdata = Console.ReadLine();
```

```

        double doubdata = double.Parse(strdata);
        Console.WriteLine("intdata = {0}; doubdata = {1}",
            intdata, doubdata);
        Console.ReadKey();
    }
}

```

*Звужуюче перетворення* - значення одного типу перетвориться до значення іншого типу, яке має менший розмір (з 64-розрядного в 32-розрядне).

Таке перетворення потенційно небезпечно втратою значення.

Звужуючі перетворення можуть приводити до втрати інформації. Якщо тип, до якого здійснюється перетворення, не може правильно передати значення джерела, то результат перетворення виявляється рівний константі

PositiveInfinity або NegativeInfinity.

При цьому значення PositiveInfinity інтерпретується як результат ділення позитивного числа на нуль, а значення NegativeInfinity інтерпретується як результат ділення негативного числа на нуль.

Якщо звужуюче перетворення забезпечується методами класу System.Convert то втрата інформації супроводиться генерацією виключної ситуації.

#### **Явне і неявне перетворення типів**

Приклади явного і неявного перетворення типів

```

int x = 25;
long y = x; // неявне
short z = (short)x; // явне
double d = 3.141592536;
float f = (float)d; // явне
long l = (long)d; // явне

```

Система Common Type System середовища Microsoft .NET забезпечує безпечну типізацію, тобто гарантує відсутність побічних ефектів (переповнення оперативної пам'яті комп'ютера, некоректне перетворення типів і т.д.).

Коли користувач вводить дані різних типів (з консолі чи форми), то він задає ці дані як рядки тексту. Дані строкового типу потребують явного перетворення до арифметичного типу.

Класи бібліотеки FCL надають два способи явного виконання таких перетворень:

Метод Parse;

Методи класу Convert.

Всі скалярні типи (арифметичний, логічний, символічний) мають статичний метод Parse, аргументом якого є рядок, а результатом об'єкт відповідного типу.

Розглянемо приклад перетворення даних з використанням методу Parse.

#### **Приклад 2.**

```

static void InputVars()
{
    string strInput;
    Console.WriteLine(INPUT_BYTE);
    strInput = Console.ReadLine();
    byte b1;
    b1 = byte.Parse(strInput);

    Console.WriteLine(INPUT_INT);
    strInput = Console.ReadLine();
    int n;
    n = int.Parse(strInput);
}

```



```

        Console.WriteLine(INPUT_FLOAT);
        strInput = Console.ReadLine();
        float x;
        x = float.Parse(strInput);

        Console.WriteLine(INPUT_CHAR);
        strInput = Console.ReadLine();
        char ch;
        ch = char.Parse(strInput);
    }

```

Перетворення в тип string завжди визначені, оскільки всі типи є нащадками базового класу object і успадковують метод ToString().

Для всіх підтипів арифметичного типу метод ToString() повертає рядок, який задає відповідне значення арифметичного типу. Метод ToString завжди потрібно викликати явно. Його можна опускати при виконанні операції конкатенації. Якщо один з операндів операції «+» є рядком, то операція сприймається як конкатенація рядків і другий операнд неявно перетворюється до цього типу. Ось відповідний приклад:

### Приклад 3.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lab2_3
{
    class Program
    {
        /// <summary>
        /// Демонстрація перетворення в рядок
        /// даних різного типу
        /// </summary>
        static void Main(string[] args)
        {
            string name;
            int age;
            double salary;
            name = "Василь Іванов";
            age = 20;
            salary = 2700;
            string s = "Ім'я: " + name +
                ". Вік: " + age.ToString() +
                ". Зарплата: " + salary;
            Console.WriteLine(s);
            Console.ReadKey();
        }
    }
}

```

Тут для змінної age метод був викликаний явно, а для змінної salary він викликається автоматично.

Для перетворень усередині арифметичного типу можна використовувати *кастинг* - *приведення* типу. Для перетворень строкового типу в скалярний тип можна використовувати метод Parse, а у зворотний бік – метод ToString.

У всіх ситуаціях, коли потрібно виконати перетворення з одного базового вбудованого типу в інший базовий тип, можна використовувати методи класу **Convert** бібліотеки FCL,

вбудованого в простір імен System, - універсального класу, статичні методи якого спеціально спроектовані для виконання перетворень.

Серед інших методів класу Convert є загальний статичний метод ChangeType, що дозволяє перетворення об'єкту до деякого заданого типу. Існує також можливість перетворення у системний тип DateTime, який хоча і не є базисним типом мови C#, але допустимий в програмах, як і будь-який інший системний тип.

Методи класу Convert підтримують загальний спосіб виконання перетворень між типами. Клас Convert містить 15 статичних методів вигляду To<Type> (ToBoolean(), ToUInt64()), де Type може набувати значень від Boolean до UInt64 для всіх вбудованих типів. Єдиним виключенням є тип object, – методу ToObject немає із зрозумілих причин, оскільки для всіх типів існує неявне перетворення до типа object.

#### Приклад 4.

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

namespace Lab1_2
{
    class Program
    {
        /// <summary>
        /// Тестування методів класу Convert
        /// </summary>

        static void Main(string[] args)
        {
            string s;
            byte b;
            int n;
            double x;
            bool flag;
            char sym;
            DateTime dt;
            sym = '7';
            s = Convert.ToString(sym);
            x = Convert.ToDouble(s);
            n = Convert.ToInt32(x);
            b = Convert.ToByte(n);
            flag = Convert.ToBoolean(b);
            x = Convert.ToDouble(flag);
            s = Convert.ToString(flag);
            s = "300";
            n = Convert.ToInt32(s);
            s = "14.09";
            s = "14.09.2008";
            dt = Convert.ToDateTime(s);
        }
    }
}
```

Цей приклад демонструє різні перетворення між типами. Всі ці перетворення виконуються явно з використанням методів класу Convert. Спочатку дані символічного типу перетворюються в рядок. Потім ці дані перетворюються в дійсний тип, потім проводяться перетворення усередині арифметичного типа з пониженням типа від double до byte. Перетворенням, що завершує приклад, є перетворення даних строкового типа до типа DateTime.

На теперішній час Microsoft рекомендує користуватися нотацією *camel* для імен змінних згідно з якою перша літера повинна бути на нижньому регістрі, наприклад: sName.

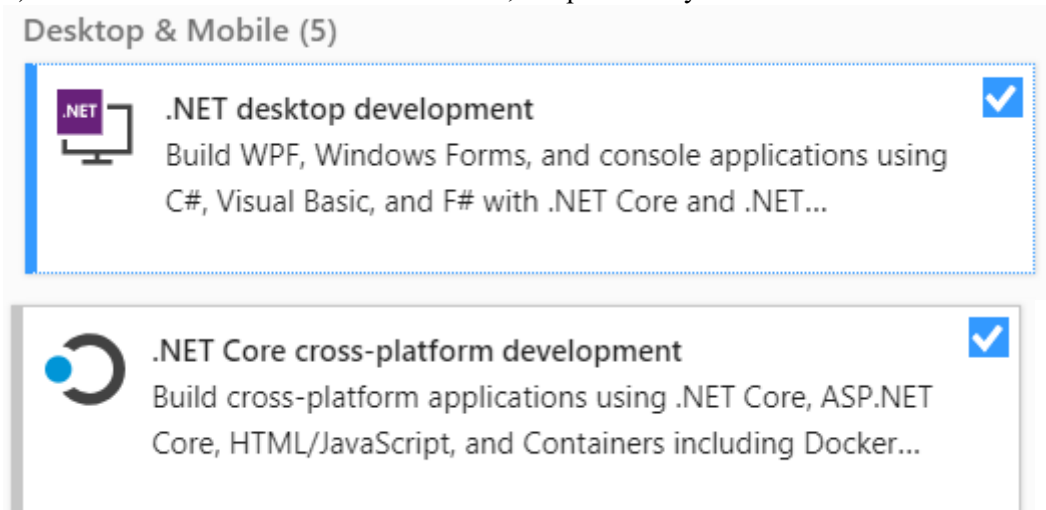
Для імен *методів і інших ідентифікаторів* рекомендується нотація Pascal за якою перша літера повинна бути на верхньому регістрі, наприклад, MyMethod.

Ідентифікатори не повинні вступати в конфлікт з ключовими словами. Крім того, вони чутливі до регістру символів, наприклад, імена sName і Sname вважаються різними.

## Хід роботи:

### 1. Підготовчий етап

1) Встановити **visual studio 2019**, вибрати наступне:



2) Встановити **git** для команд в консолі.

3) Встановити розширення **.Net Core** для доповнених команд.

2. Зареєструватись на github (Якщо уже маєте власний аккаунт, можете використовувати його).

<https://github.com>

3. Створити репозиторій.

#### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner: rip9991 /  ✓

Great repository names are short and memorable. Need inspiration? How about [musical-garbanzo?](#)

Description (optional)

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

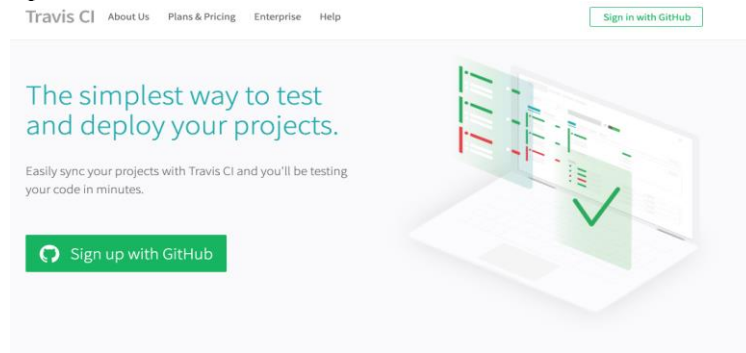
☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** ⓘ

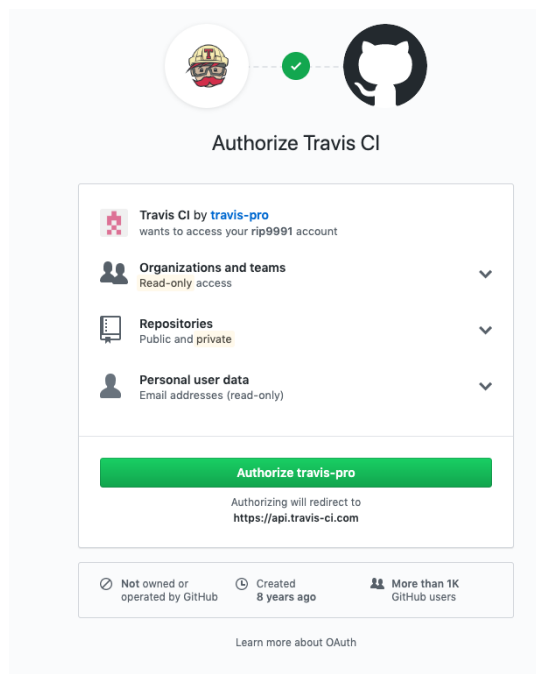
Create repository

4. Перейти на сайт <https://travis-ci.com> та авторизуватись через github аккаунт.

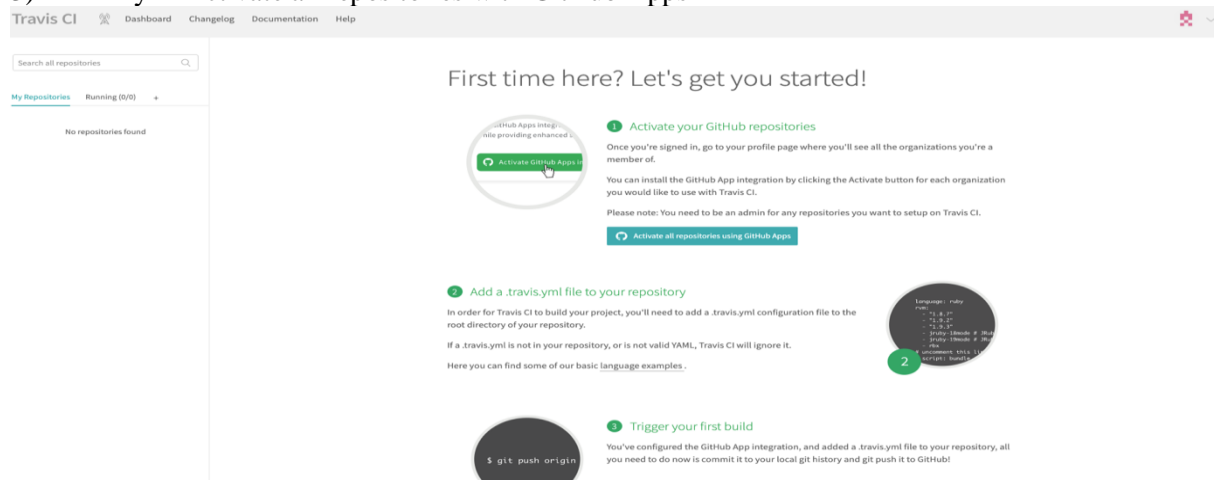
1) натиснути Sign up with GitHub



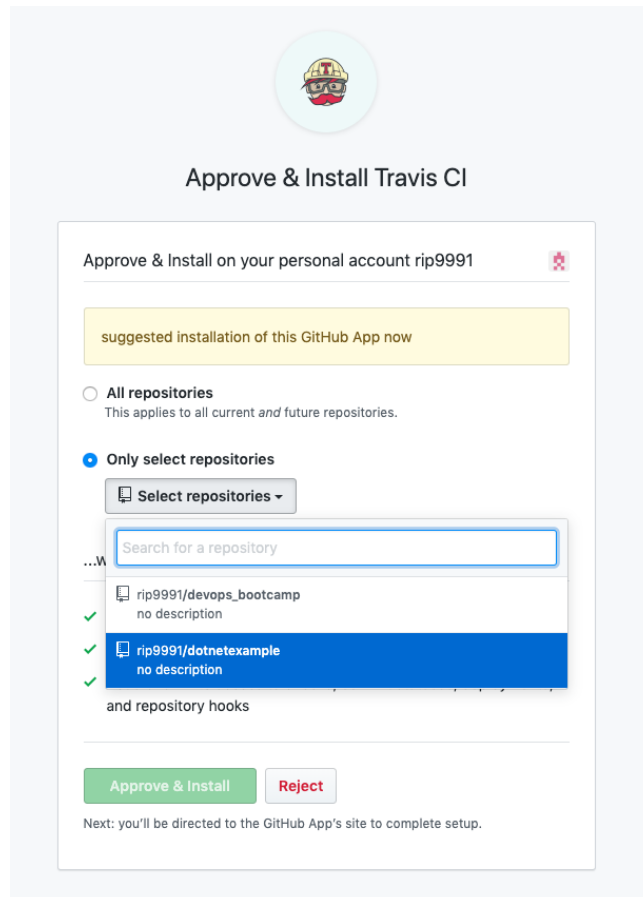
2) Натиснути authorize travis-pro



3) Натиснути Activate all repositories with GitHub Apps

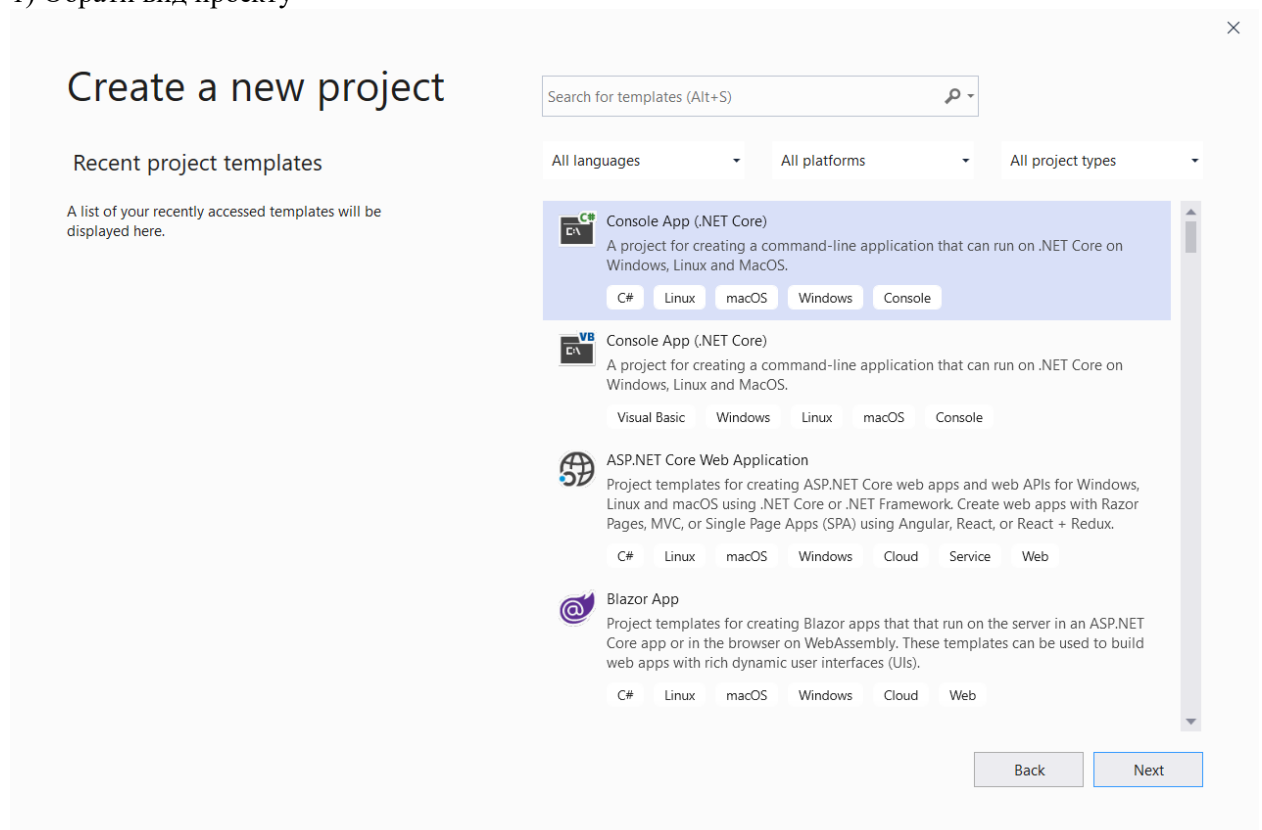


4) Вибрати потрібний репозиторій та натиснути Approve & Install щоб Travis підв'язався до репозиторію

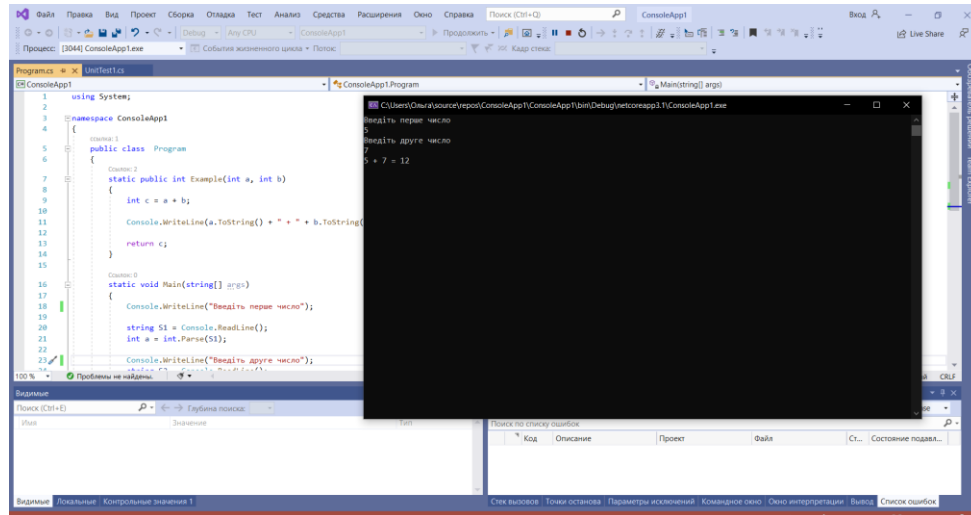


5. Відкрити Visual Studio та створити простий консольний проект.

1) Обрати вид проекту



2) Створити та звиконати проект, що розв'язує задачу пошуку суми двох чисел, заданих користувачем.

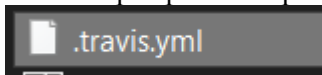


Використайте наступний код:

```
using System;
namespace ConsoleApp1
{
    public class Program
    {
        static public int Example(int a, int b)
        {
            int c = a + b;
            Console.WriteLine(a.ToString() + " + " + b.ToString() + " = " +
c.ToString());
            return c;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Введіть перше число");
            string S1 = Console.ReadLine();
            int a = int.Parse(S1);
            Console.WriteLine("Введіть друге число");
            string S2 = Console.ReadLine();
            int b = int.Parse(S2);
            Example(a, b);
            Console.ReadLine();
        }
    }
}
```

6. Зберегти проект, та створити у папці проекту файл .travis.yml щоб з допомогою функціоналу Travis перевірити та протестувати проект.



Та вписати в нього наступний код:

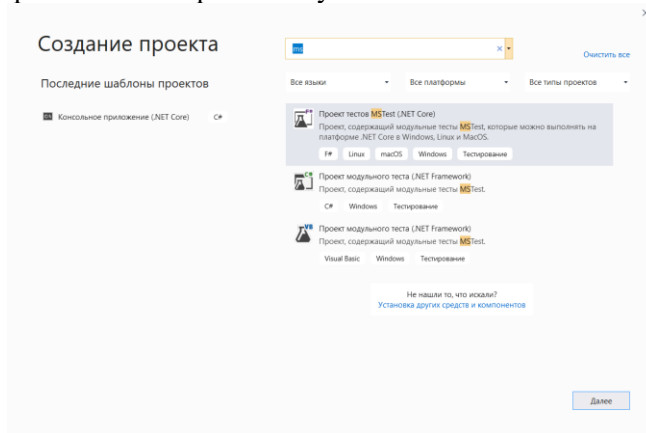
```
language: csharp
mono: none
dotnet: 3.1
solution: ConsoleApp1.sln
script:
  - dotnet restore
  - dotnet test

//Виконавчий файл (заміняєте іменем вашого файлу )
//
//два пробіла перед командою обов'язково
//два пробіла перед командою обов'язково
```

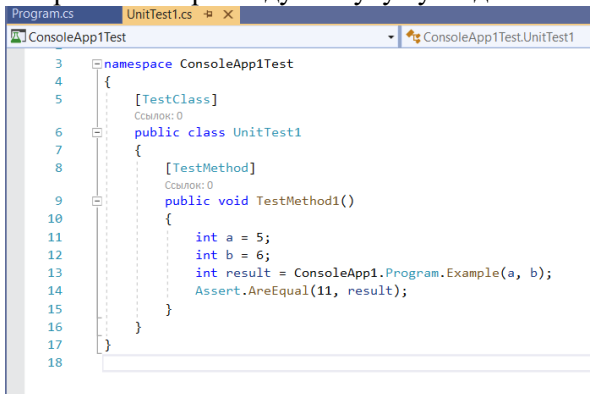
```
1 language: csharp
2 mono: none
3 dotnet: 3.1
4 solution: ConsoleApp1.sln
5 script:
6   - dotnet restore
7   - dotnet test
```

## 7. Створюєте тест до проекту.

1) Для цього створюєте новий проект типу MSTest



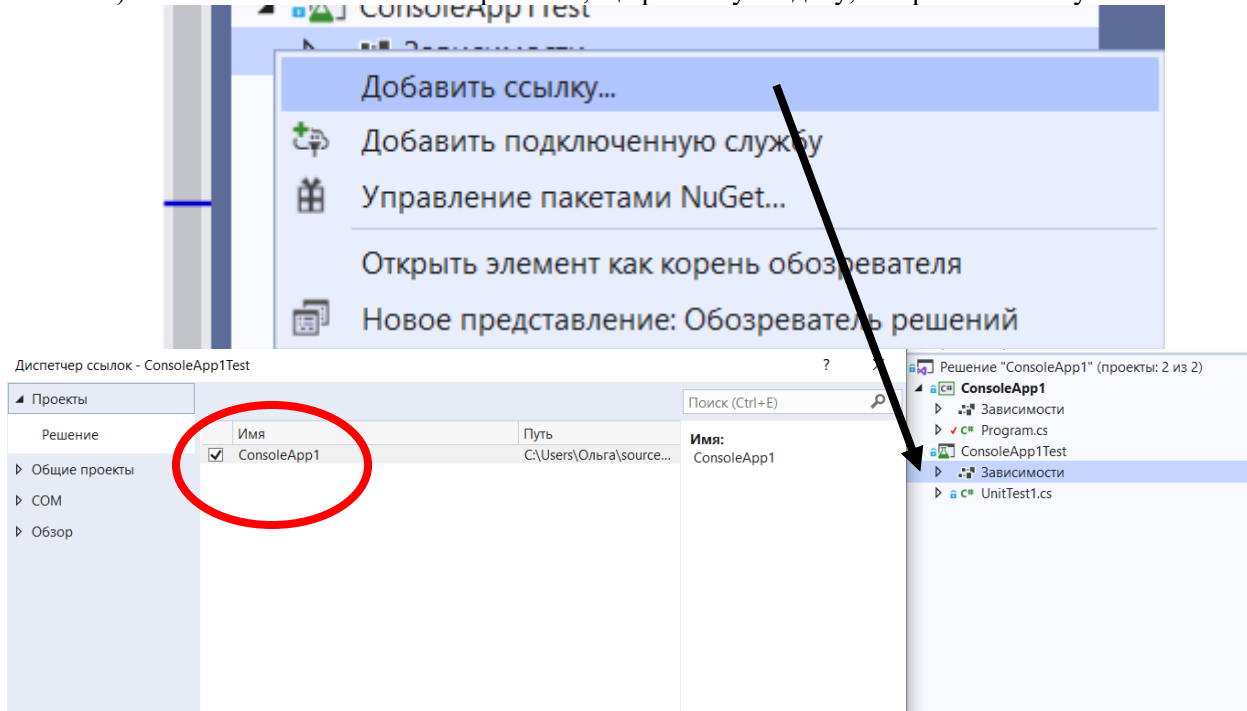
2) Створюєте тест до контрольного прикладу пошуку суми двох чисел наступного виду:



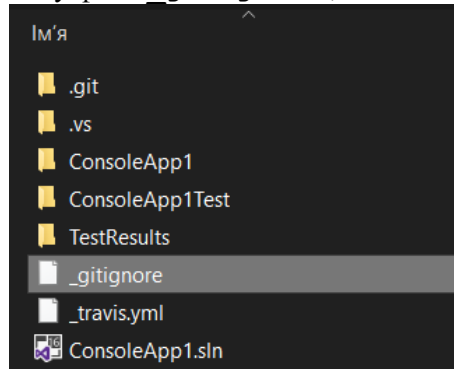
Вхідні дані: перше число 5, друге число 6

*Вихідні дані:* сума чисел 11

3) Встановлюєте зв'язок між проектом, що розв'язує задачу, та проектом тесту

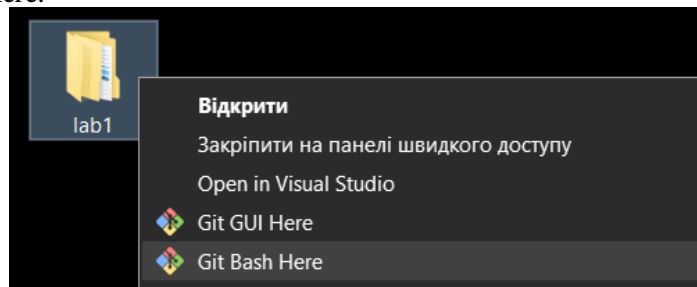


- 4) Перевіряєте коректність роботи тесту
- 5) Додайте до папки проекту файл **gitignore**, який додано до методичних матеріалів.



8. Завантажити папку з проектом до свого репозиторію на GitHub.

- 1) Щоб відкрити консоль git потрібно ПКМ по папці, вміст якої хочемо перекинути та натиснути Git Bash Here.



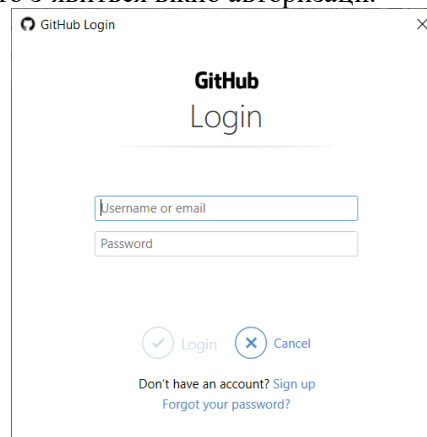
- 2) Щоб імпортувати проект в ваш git потрібно:
  - а) Вказати папку яку хочемо перенести;  
`$ git init` (вибирається та папка через яку ми відкрили консоль)
  - б) Вибрати файли, які потрібно перенести; `$ git add .`
  - в) Створити збереження (*commit*) і дати назву  
`$ git commit -m "first test"`
  - г) Вказати власний репозиторій з допомогою команди  
`$ git remote add origin`

Наприклад, `$ git remote add origin https://github.com/Olgaarte/lab01.git`

(Якщо з'являється помилка «fatal: remote origin already exists.»  
переходьте до наступного кроку.)

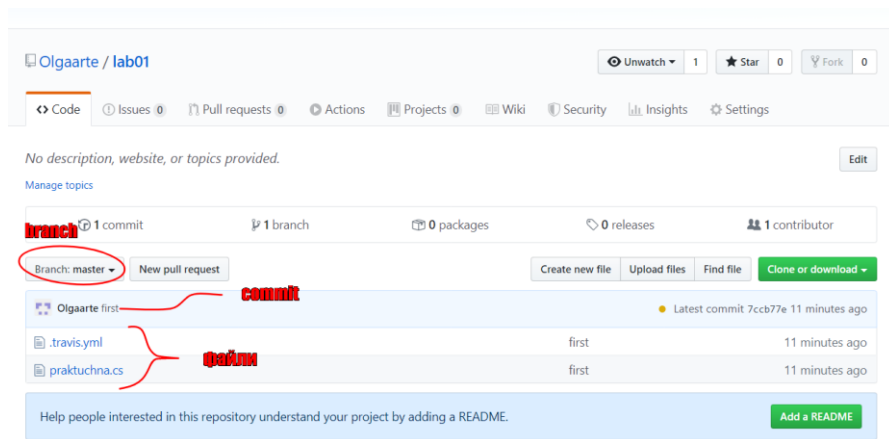
- д) Відправити файл(и) вказавши вітку (branch)  
`$ git push -u origin master`  
 (master – перша стандартна вітка)

Якщо все зроблено правильно, то з'явиться вікно авторизації.



Після проходження авторизації ви повинні побачити додані файли свого проекту до відповідної гілки, наприклад:





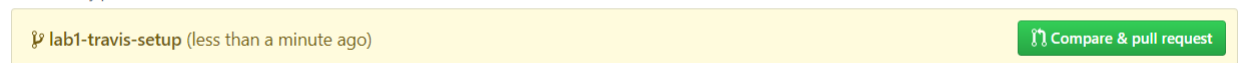
Отже, ваш проект успішно завантажений на Git Hub.

## !!!Корисне посилання для вивчення базових команд та роботи з Git Hub !!!

<https://learngitbranching.js.org/>

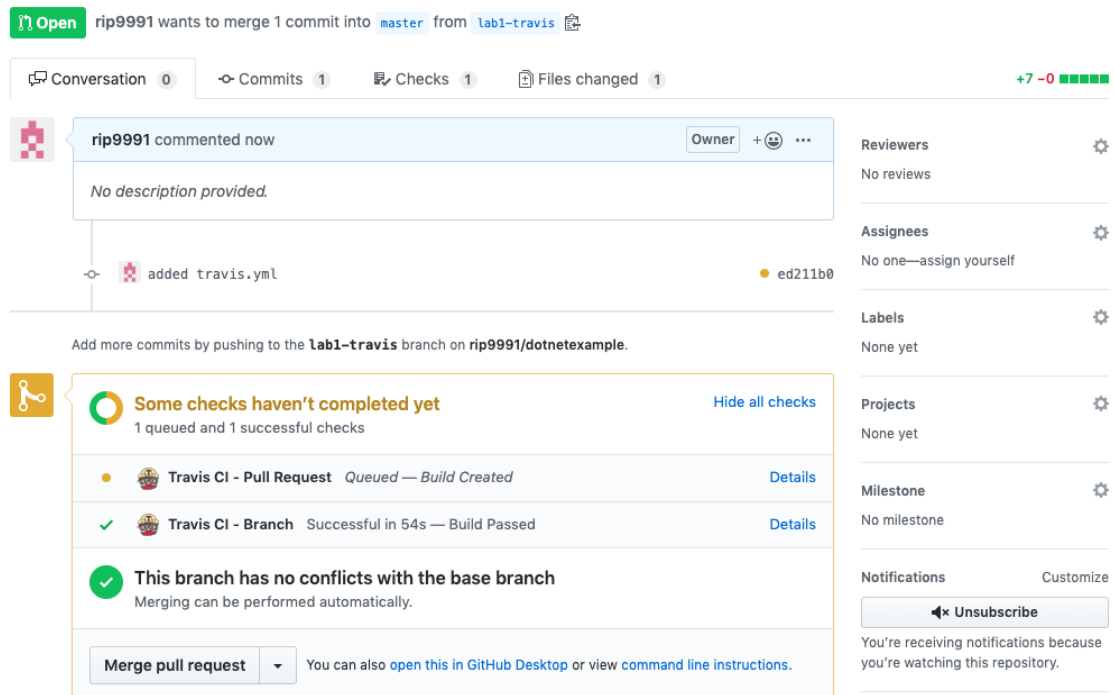
9. Перейти на сторінку Travis CI та створити pull request – натиснути **Compare & Pull Request** (Create new Request). В compare вказати потрібну брэнчу. В Base залишити мастер. Написати назву, додати комент та натиснути **Create Pull Request**

Your recently pushed branches:



Процес Тревіс білд автоматично запуститься

### added travis.yml #1




10. Після успішного білда **All checks have passed** має з'явитись. Натиснути **merge pull request** і код попаде в мастер брэнчу (1 бал)

## added travis.yml #1

[Edit](#)[Open](#) rip9991 wants to merge 1 commit into master from lab1-travis


Conversation 0 Commits 1 Checks 2 Files changed 1

+7 -0




rip9991 commented 1 minute ago

No description provided.

 added travis.yml ✓ ed211b0

Add more commits by pushing to the lab1-travis branch on rip9991/dotnetexample.



✓ All checks have passed

2 successful checks

Show all checks

✓ This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Customize

Unsubscribe

11. Створити нову вітку (бренчу) для виконання індивідуального завдання, наприклад `git checkout -b lab1-mywork`

12. Виконати індивідуальне завдання лабораторної роботи (1 бал)

13. Додати тести до індивідуального завдання в тест проект. Приклад проекту з тестами є вище (1 бал).

14. Завантажити власний проект та його тести в гіт та виконати pull request (1 бал)

Після того як тести пройшли успішно змерджити в мастер. Лог виконання тестів можна переглянути в <https://travis-ci.com/dashboard>

**Корисні команди для роботи з git:**

1. Клонувати репозиторій  
`$ git clone`
2. Скачати останню версію коду  
`git checkout master`  
`git pull origin master`

### Індивідуальні завдання:

(Для виконання індивідуальних завдань № варіанта є порядковим номером прізвища студента в списку групи)

**Написати програму, що розв'язує задачу, згідно номера свого варіанта. Для написання тесту використати дані контрольного прикладу до свого варіанта:**

1. Дано два числа  $a$  та  $b$ . Знайти їх середнє арифметичне та середнє геометричне.

Вхідні дані:  $a = 3, b = 27$ .

Вихідні дані:  $sa = 15, sg = 9$ .

2. З початку доби пройшло  $N$  секунд ( $N$  - ціле). Знайти кількість повних хвилин, що пройшли з початку доби.

Вхідні дані:  $N = 550$ .

Вихідні дані:  $m = 9$ .

3. Дано  $a$  та  $b$  – катети прямокутного трикутника. Знайти гіпотенузу  $c$  та його периметр.

Вхідні дані:  $a = 3, b = 4$ .

Вихідні дані:  $c = 5, p = 12$ .

4. Дано довжини ребер прямокутного паралелепіпеда  $a$ ,  $b$  та  $c$ . Знайти його об'єм та площу поверхні.  
*Вхідні дані:*  $a = 1, b = 2, c = 3$ .  
*Вихідні дані:*  $V = 6, S = 22$ .
5. Дано координати трьох вершин трикутника  $(x_1, y_1)$ ,  $(x_2, y_2)$  та  $(x_3, y_3)$ . Використовуючи формулу обчислення відстані між двома точками на площині знайти його периметр та площу.  
*Вхідні дані:*  $x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 3, x_3 = 4, y_3 = 0$ .  
*Вихідні дані:*  $P = 12, S = 6$ .
6. Дано циліндр  $R$ . Знайти площу його бічної поверхні та об'єм.  
*Вхідні дані:*  $r = 3, h = 10$ .  
*Вихідні дані:*  $S = 188,4, V = 244,92$ .
7. Дано трьохзначне число. Вивести першу та останню цифру даного числа.  
*Вхідні дані:*  $d = 357$ .  
*Вихідні дані:*  $d1 = 3, d2 = 7$ .
8. Дано чотиризначне число. Визначити суму та добуток його цифр.  
*Вхідні дані:*  $d = 1352$ .  
*Вихідні дані:*  $S = 11, D = 30$ .
9. Дано трьохзначне число  $x$ . В ньому закреслили його останню цифру. Після цього цифри, які залишилися поміняли місцями і зліва до них дописали закреслену цифру числа  $x$ . Отримали число 654. Знайти число  $x$ .  
*Вхідні дані:*  $d = 654$ .  
*Вихідні дані:*  $x = 456$ .
10. З початку доби пройшло  $n$  секунд. Визначити скільки годин, хвилин та секунд пройшло з початку доби. Вивести інформацію в наступному вигляді: "З початку доби пройшло ... годин ... хвилин ... секунд".  
*Вхідні дані:*  $n = 3680$ .  
*Вихідні дані:* "З початку доби пройшло 1 годин 1 хвилин 20 секунд".
11. Дано значення змінних  $A, B, C$ . Змінити їх значення, перемістивши значення  $A$  в  $B$ ,  $B$  – в  $C$ ,  $C$  – в  $A$ .  
*Вхідні дані:*  $A = 4, B = 5, C = 6$ .  
*Вихідні дані:*  $A = 6, B = 4, C = 5$ .
12. Знайти значення функції  $y = x^3 - 4x^2 - 5x + 9 + \cos x$  при заданому значенні  $x$ .  
*Вхідні дані:*  $x = 0$ .  
*Вихідні дані:*  $y = 10$ .
13. Дано  $a$  довжину ребра куба. Знайти об'єм куба та площу його поверхні.  
*Вхідні дані:*  $a = 4$ .  
*Вихідні дані:*  $V = 64, S = 96$ .
14. Дано два числа  $a$  та  $b$ . Знайти їх середнє арифметичне та середнє геометричне.  
*Вхідні дані:*  $a = 3, b = 27$ .  
*Вихідні дані:*  $sa = 15, sg = 9$ .
15. З початку доби пройшло  $N$  секунд ( $N$  – ціле). Знайти кількість повних хвилин, що пройшли з початку доби.  
*Вхідні дані:*  $N = 550$ .  
*Вихідні дані:*  $m = 9$ .
16. Дано  $a$  та  $b$  – катети прямокутного трикутника. Знайти гіпотенузу  $c$  та його периметр.  
*Вхідні дані:*  $a = 3, b = 4$ .  
*Вихідні дані:*  $c = 5, p = 12$ .

17. Дано довжини ребер прямокутного паралелепіпеда  $a$ ,  $b$  та  $c$ . Знайти його об'єм та площу поверхні.

Вхідні дані:  $a = 1, b = 2, c = 3$ .

Вихідні дані:  $V = 6, S = 22$ .

18. Дано координати трьох вершин трикутника  $(x_1, y_1)$ ,  $(x_2, y_2)$  та  $(x_3, y_3)$ . Використовуючи формулу обчислення відстані між двома точками на площині знайти його периметр та площу.

Вхідні дані:  $x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 3, x_3 = 4, y_3 = 0$ .

Вихідні дані:  $P = 12, S = 6$ .

19. Дано циліндр  $R$ . Знайти площу його бічної поверхні та об'єм.

Вхідні дані:  $r = 3, h = 10$ .

Вихідні дані:  $S = 188,4, V = 244,92$ .

20. Дано трьохзначне число. Вивести першу та останню цифру даного числа.

Вхідні дані:  $d = 357$ .

Вихідні дані:  $d1 = 3, d2 = 7$ .

21. Дано чотиризначне число. Визначити суму та добуток його цифр.

Вхідні дані:  $d = 1352$ .

Вихідні дані:  $S = 11, D = 30$ .

22. Дано трьохзначне число  $x$ . В ньому закреслили його останню цифру. Після цього цифри, які залишилися поміняли місцями і зліва до них дописали закреслену цифру числа  $x$ . Отримали число 654. Знайти число  $x$ .

Вхідні дані:  $d = 654$ .

Вихідні дані:  $x = 456$ .

23. З початку доби пройшло  $n$  секунд. Визначити скільки годин, хвилин та секунд пройшло з початку доби. Вивести інформацію в наступному вигляді: "З початку доби пройшло ... годин ... хвилин ... секунд".

Вхідні дані:  $n = 3680$ .

Вихідні дані: "З початку доби пройшло 1 годин 1 хвилин 20 секунд".

24. Знайти значення функції  $y = x^3 - 4x^2 - 5x + 9 + \cos x$  при заданому значенні  $x$ .

Вхідні дані:  $x = 0$ .

Вихідні дані:  $y = 10$ .

25. Дано  $a$  довжину ребра куба. Знайти об'єм куба та площу його поверхні.

Вхідні дані:  $a = 4$ .

Вихідні дані:  $V = 64, S = 96$ .

Контрольні запитання (1 бал):

1. Перерахуйте і опишіть лексеми мови C#.
2. Перерахуйте і опишіть літерали мови C#.
3. Перерахуйте вбудовані типи даних і опишіть їх.
4. Які елементи стандартних класів .NET, відповідних вбудованим типам мови, ви знаєте?
5. Де можна описувати змінні? Що входить в опис змінної?
6. Що відбувається при використанні у виразі операндів різних типів? Наведіть приклади.
7. Які методи використовуються для перетворення даних з типу string в арифметичні типи?
8. Який метод використовується для перетворення з арифметичного типу в тип string ?