



ОСНОВИ ПРОГРАМУВАННЯ НА МОВІ С++

Урок № 1

Вступ у мову програмування C++

Зміст

1. Попередні міркування	4
Вступне слово	4
Історичні факти.....	5
2. Перше знайомство з алгоритмами	10
Поняття алгоритму	10
Властивості алгоритму	11
Типи алгоритмів:	
лінійний, розгалужений, циклічний.....	13
3. Графічний спосіб опису алгоритму	15
Поняття блок-схеми	15
Базове позначення в блок-схемах.....	15
4. Інсталяція Microsoft Visual Studio 2019	20
5. Перший проект	28
Ситуація перша. Команди написані інтерпретованою мовою	29

Ситуація друга. Команди написані компільованою мовою.....	30
Проба пера.....	31
Приклад першої програми на мові C++.....	35
Відкриття збереженого проекту	38
6. Вивід даних	40
P. S.	45
7. Типи даних	47
Числові типи	48
Символьний тип	49
Логічний тип	50
8. Змінні константи	51
Правила складання імен	52
Оголошення та використання змінних і констант....	53
Вивід значення змінної на екран	53
Практичні приклади	54
9. Ввід даних.....	60
10. Літерали	64
11. Домашнє завдання	65

Додаткові матеріали уроку прикріплені до даного PDF-файлу.
Для доступу до матеріалів, урок необхідно відкрити в програмі
Adobe Acrobat Reader.

1. Попередній міркування

Вступне слово

Ласкаво просимо у світ програмування! Що таке програмування? Напевно, кожен із вас чув це слово. Якщо перефразувати відому цитату, можна сказати так: «Програміст — це звучить гордо!» І це дійсно так. Якщо ви натрапляли в інтернеті на сайти з пропозиціями роботи, то звертали увагу на розмір заробітної плати, яка пропонується програмістам. Звичайно, виникає логічне питання — чому такий високий рівень оплати праці програміста? Усе згідно із законами ринку: є попит на програмістів, але на даний момент кількість хороших фахівців дуже мала. Звісно, якби програмістом можна було стати за короткий термін, завчивши низку слів, професія навряд чи була б такою популярною. Але не засмучуйтесь! Ми, ваші викладачі, постараємося зробити все від нас залежне для того, щоб не тільки розповісти вам як програмувати, а й навчити вас жити програмуючи. Однак наша з вами спільна робота має бути взаємною. Викладач, хоч би як він стався, не зможе навчити студента, якщо останній цього не захоче. Робота, тільки постійна робота над собою зможе провести вас до вершини майстерності.

Щиро сподіваємося на те, що ви захопитеся чарівним світом програмування!

Перш ніж приступати до вивчення будь-якої науки, завжди необхідно з'ясувати, звідки ця наука з'явилася, як вона розвивалася, які її коріння і значення в історії.

Історичні факти

Англійський математик XIX століття Шенкс витратив понад 20 років свого життя на обчислення числа Пі з точністю до 707 значущих цифр після коми. Цей результат отримав славу рекорду обчислень XIX століття. Однак згодом було виявлено, що Шенкс помилився в 520-му знаку, і тому всі наступні значущі цифри були обчислені невірно.

У 1804 році французький винахідник Жозеф Мари Жаккар створив «програмно-керований» ткацький верстат. Для керування верстатом використовувалися перфокарти, з'єднані одна з одною у вигляді стрічки. Дерев'яні шпильки «читаючого пристрою» верстата за розташуванням отворів у перфокарті визначали, які нитки слід підняти, а які опустити для отримання потрібного візерунка. У 1890 році в США винахідник Герман Голлеріт розробив електромеханічну лічильну машину — табулятор, керований перфокартами. Вона була використана для складання таблиць із результатами перепису населення США. Заснована Голлерітом фірма з виробництва табуляторів згодом перетворилася на корпорацію *International Business Machines* (IBM).

У 1936 році двадцятип'ятирічний студент Кембріджського університету англієць Алан Тюрінг опублікував статтю «Про обчислення числа», у якій розглядався гіпотетичний пристрій («машина Тюрінга»), придатний для вирішення будь-якої математичної або логічної задачі, яку можливо розв'язати, — прообраз програмованого комп'ютера.

У 1941 році німецький інженер Конрад Цузе побудував діючий комп'ютер Z3, у якому використовувалася

двійкова система числення. Програми записувалися на перфострічці.

У 1945 році у вищому технічному училищі Пенсільванського університету (США) фізик Джон Моклі та інженер Джон Преспер Екерт побудували повністю електронну машину ENIAC. Аби задати програму, було необхідно вручну встановити тисячі перемикачів, увіткнути сотні штекерів у гнізда контактної панелі. 1 червня 1945 року був розісланий звіт американського математика угорського походження Джона фон Неймана

«Попередній звіт по EDVAC», що містить концепцію зберігання команд комп'ютера в його власній внутрішній пам'яті.

21 червня 1948 року в Манчестерському університеті (Великобританія) на машині «Марк-1» виконана перша у світі програма, що зберігається в пам'яті машини, — пошук найбільшого співмножника заданого числа.

У 1949 році під керівництвом Моріса Вілкса був створений комп'ютер EDSAC. Проектувальники EDSAC ввели систему мнемонічних позначень, де кожна машинна команда представлялася однією великою літерою, та автоматизували налаштування підпрограм на певне місце в пам'яті. Моріс Вілкс назвав мнемонічну схему й бібліотеку підпрограм «збираюча система» (*assembly system*) — звідси слово «асемблер».

У 1949 році у Філадельфії (США) під керівництвом Джона Моклі був створений «Короткий код» — перший примітивний інтерпретатор мови програмування.

У 1951 році у фірмі Remington Rand американська програмістка Ірейс Гоппер розробила першу транслюючу

програму. Гоппер назвала її компілятором (*compiler* — компонувальник).

У 1957 році на 20-му поверсі штаб-квартири фірми IBM на Медісон-авеню в Нью-Йорку народилася мова Фортран (*FORmula TRANslation* — трансляція формул). Групою розробників керував 30-річний математик Джон Бекус. Фортран — перша зі «справжніх» мов високого рівня.

У 1963 році була створена мова програмування Бейсик. Засновниками мови стали Джон Кемені і Томас Курц, співробітники Дартмут Коледжу. Під керівництвом творців мова була реалізована групою студентів коледжу. Найперший діалект мови називався Dartmouth BASIC.

У 1958-1968 роках велася розробка та удосконалення мови програмування під назвою АЛГОЛ, назва якої походить від словосполучення «алгоритмічна мова» (*algorithmic language*). На відміну від Фортрана, який переважно використовувався в США й Канаді, АЛГОЛ був широко розповсюджений у Європі та СРСР. Мова була створена міжнародним комітетом, до складу якого входили європейські та американські вчені — Джон Бекус, Пітер Наур, Воллі Фойрцойг, Ніклаус Вірт.

У 1970 Ніклаус Вірт створив мову програмування, назву якій дав на честь французького фізика й математика Блеза Паскаля. Паскаль планувався Віртом, як мова, що навчає процедурному програмуванню. У 1972 році 31-річний фахівець з системного програмування, який працював у фірмі Bell Labs, Денніс Рітчі розробив мову програмування Сі.

Перший опис мови був представлений у книзі Б. Кернігана і Д. Рітчі, яку було перекладено на українську.

Тривалий час цей опис був стандартом, проте низка моментів допускала неоднозначне тлумачення, яке породило безліч трактувань мови Сі. Для виправлення цієї ситуації при Американському інституті національних стандартів (ANSI) був створений комітет зі стандартизації мови Сі.

У 1983 році був затверджений стандарт мови Сі, який отримав назву ANSI C.

На початку 80-х років у Bell Laboratory Б'янн Страуструп, у результаті доповнення й розширення мови Сі, створив нову по суті мову, що отримала назву «Сі з класами». У 1983 році ця назва була замінена на С++.

23 травня 1995 року компанія Sun Microsystems випустила нову мову програмування під назвою Oak. Мова була розроблена для програмування побутової електроніки. Надалі Oak був перейменований на мову Java і почав широко використовуватися в розробці додатків і серверного програмного забезпечення.

У 2000-2001 роках була прийнята і стандартизована нова мова програмування С# (Сі-шарп), спеціально розроблена для платформи .NET. У створенні мови брали участь співробітники Microsoft Research (НДІ при корпорації Microsoft) — Андерс Гейлсберг, Скот Вілтамут, Пітер Гольде та інші відомі фахівці, озкрема Ерік Меер. Версія мови — С# 2.0 була представлена навесні 2005 року. Варто зазначити, що однією з мов, що слугували базою для С#, був старий-добрий С++.

Після того, як ми зробили невелику подорож у минулому, ми можемо переходити до сьогодення. Для подальшої роботи нам з вами безумовно знадобиться програмне забезпечення. У зв'язку зі швидким розвитком ринку

ІТ-технологій постійно створюються більш нові версії різних програм. У наступних розділах уроку ми розглянемо основи побудови алгоритмів та інсталяцію програмного продукту, який ми будемо використовувати у межах процесу навчання.

2. Перше знайомство з алгоритмами

Поняття алгоритму

Щодня ми стикаємося з рішенням простих повсякденних завдань. Як встигнути все зробити? Ми плануємо кожну хвилину протягом дня, щоб найбільш вигідно використати власний час. Виставляємо пріоритети, визначаємо послідовність виконання своїх дій. І все це робимо заради того, щоб нарешті отримати потрібний нам результат. Виявляється, що ми постійно стикаємося з інструкціями, приписами, рецептами, правилами, відповідно до яких відбувається та чи інша людська діяльність.

Часто виконавцем алгоритму є комп'ютер, але це не завжди так. Уявімо просту ситуацію — приготування сніданку. У цій ситуації виконавцем є людина. Які будуть її дії, якщо необхідно приготувати яечню?

1. Включити плиту.
2. Поставити пательню.
3. Налити на неї масло.
4. Розігріти пательню.
5. Розбити 2 яйця.
6. Смажити 5 хвилин на повільному вогні.

При виконанні такої послідовності дій, отримаємо гарячий і смачний сніданок. А що буде, якщо ми випадково поміняємо місцями другий і третій пункти? Це приведе до брудної плити й поганого настрою на весь день. Отже,

алгоритм — це чітка і строга послідовність дій, яка веде до потрібного результату. Уявімо собі роботу великого заводу з виробництва машин на продаж. Зауважимо, що потрібний результат у цьому випадку є автомобіль і в жодному разі брак. Адже важлива не сама наявність результату, а його якість — автомобіль, що працює правильно. В іншому випадку, можливі велики фінансові втрати для заводу. Так і в нашому випадку — невірно написаний алгоритм просто напросто буде нікому не потрібен.

Властивості алгоритму

Алгоритм створюється так, щоб виконавець зміг абсолютно точно слідувати зазначенім інструкціям та ефективно отримати необхідний результат. Тому на алгоритм лягає кілька основних вимог у вигляді його властивостей.

Результативність. Насамперед, при написанні будь-якого алгоритму ми повинні бути спрямовані на отримання потрібного результату.

Коректність. Крім цього, отриманий результат має бути правильним (готовий сніданок чи брудна плита?). Адже, коли складається сам алгоритм, можливий результат можна визначити заздалегідь. Якщо після виконання інструкцій ми все ж таки отримуємо іншу відповідь, відмінну від очікуваної, швидше за все або послідовність дій, або самі дії були невірні.

Точність. Усі наші дії мають бути вказані точно, не двояко. Інакше виконавцю такого алгоритму не буде зрозуміло, скільки часу потрібно смажити яечню або скільки яєць розбивати на пательню.

Зрозумілість. Виконавець алгоритму має повністю розуміти, які дії він змушений виконати. В іншому випадку такий алгоритм буде марним. Уявімо собі, що хтось покаже наш рецепт іноземцю. Він не зрозуміє, чого від нього хочуть, і не виконає поставлену задачу. Отже, необхідно використовувати одну мову спілкування. Для створення програм у нашому курсі ви вивчите C++, що містить свої правила використання.

Дискретність. Неможливо одночасно виконати всі дії одразу. Спочатку визначаємо, які саме дії потрібні, а після цього встановлюємо їхню послідовність. Точніше, розбиваємо велику задачу на підзадачі. Після виконання однієї такої підзадачі, приступаємо до наступної і так далі.

Масовість. Використання алгоритму мусить приводити до вирішення подібних однорідних задач. Наш рецепт можна застосувати до різних видів яєць, змінивши при цьому лише час їхнього приготування.

Усі ці властивості утворюють основу, на яку спирається алгоритм (мал. 1).



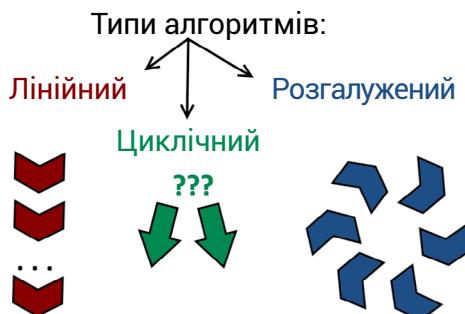
Малюнок 1

При порушенні будь-якої з цих властивостей виникає велика ймовірність створення нестабільної системи.

Типи алгоритмів: лінійний, розгалужений, циклічний

Наше життя непостійне. Залежно від ситуації нам доводиться змінювати задумані плани, змінювати послідовність наших рішень. У голові зріють питання: як вчинити в цій ситуації? який вибрати шлях? і який при цьому вийде результат?

Усі завдання, з якими ми стикаємося, неможливо виконати лише одним універсальним способом. Тому в алгоритмуванні існує три типи алгоритмів: лінійний, розгалужений і циклічний (мал. 2).



Малюнок 2

Лінійний. Виконання дій проводиться в строгому порядку. Наприклад, рецепт приготування будь-якої страви з кулінарної книги. Ми завжди знаємо, який продукт брати, у якій кількості і що з ним робити. При виконанні такої чіткої інструкції в результаті отримаємо потрібну страву.

Розгалужений. Надається вибір одного з декількох варіантів можливих шляхів. Уявімо собі ситуацію вдома. Ви лежите на дивані. За вікном йде дощ. Якщо дощ припиниться, тоді ви підете гуляти на свіжому повітрі, інакше — дивітесь телевізор. І, як бачимо, залежно від погоди на вулиці, зможемо піти тільки по одній гілці алгоритму.

Циклічний. Повтор одних і тих самих дій за заданою умовою. Розглянемо ситуацію в саду. Садівник має заповнити квітами всю клумбу. Спочатку він бере першу квітку. Після цього викопує ямку для неї. Поливає її водою. Далі поміщає в цю лунку квітку і закриває її коріння землею. Але ж це тільки одна квітка?! Отже, садівник має повторювати ці дії доти, доки він не засадить усю клумбу квітами.

У нашему курсі під час створення алгоритмів ви навчитесь правильно підбирати порядок команд і їхню структуру поведінки. Адже наявність алгоритму виключає міркування виконавця і дає можливість автоматично розв'язувати задачі комп'ютером, який виконує команди алгоритму в зазначеній послідовності.

3. Графічний спосіб опису алгоритму

Поняття блок-схеми

В одному з наукових досліджень вказується, що за сприйняттям інформації більшість людей є візуалами. Тобто аналіз даних освоюється легше не в текстовому вигляді, а саме в графічному. Кілька років тому, коли не було настільки популярне використання гаджетів, для більшості людей однією з проблем була орієнтація на місцевості. Іноді доводилося кілька разів пояснювати як дістатися потрібного місця. Зрештою, найкмітливіші діставали аркуш паперу і малювали схему проїзду. Використання стрілок і назв вулиць давало змогу вирішувати таку маленьку проблему.

Отже, графічний спосіб унікальний тим, що можна швидко визначити суть усього алгоритму. Тому для пояснення більшості алгоритмів використовують блок-схеми.

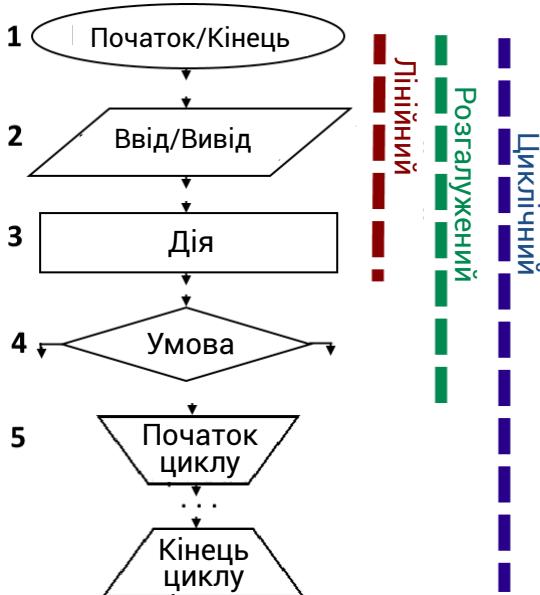
Блок-схема — графічний спосіб опису алгоритму. Вона складається з декількох функціональних блоків, кожен з яких так само виконує своє чітке призначення.

Базове позначення в блок-схемах

Для побудови блок-схем використовують такі елементи (мал. 3).

Для переходу між елементами блок-схеми використовують стрілочки. Зверніть увагу, що блоки мають різну кількість входів і виходів:

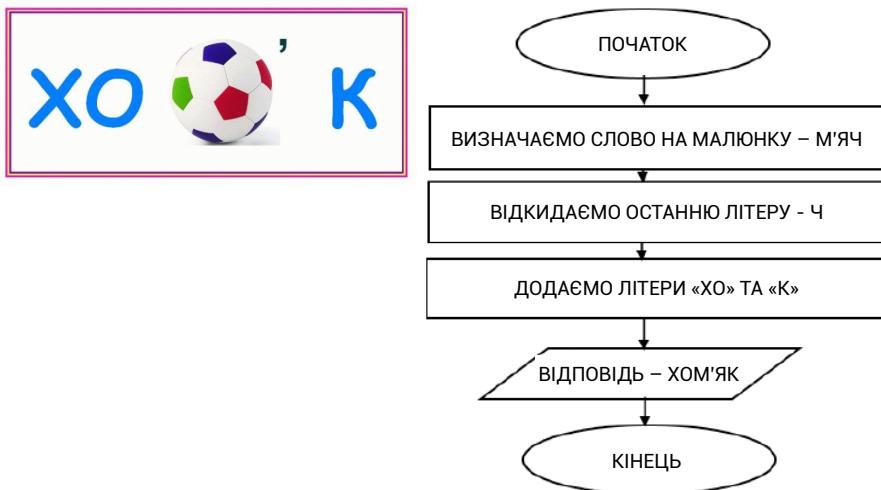
- Блок Початок — тільки вихід.
- Блок Кінець — тільки вход.
- Блоки Ввід-Вивід, Дія й Цикл — один вход та один вихід.
- Блок Умова — один вход і два виходи, причому гілки виходу прийнято підписувати (так/ні, **+/ -, true / false**).

**Малюнок 3**

Блок-схема будується строго зверху вниз. Можливе використання будь-якої кількості блоків залежно від поставленої задачі, але блок «Початок/Кінець» дозволено розміщувати лише двічі — на початку самого алгоритму й у кінці.

Усі ми в дитинстві намагалися розгадувати ребуси. Так от, ми неусвідомлено виконували простий алгоритм їхнього розгадування. Строго читали картинки зліва

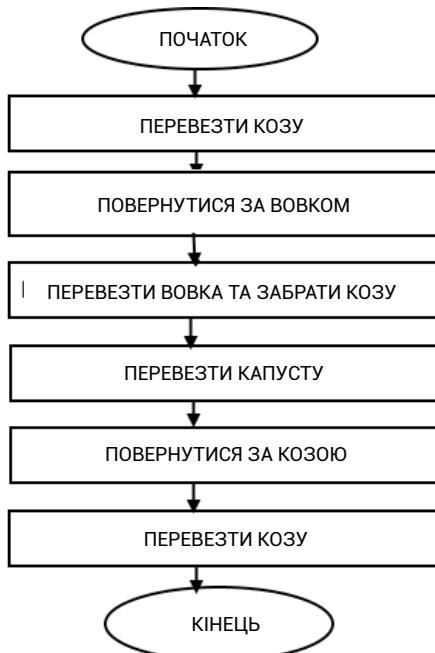
направо, де потрібно — пропускали літери, дотримувалися правил. І в результаті отримували правильну відповідь. Ось, наприклад, подивимося як розгадати такий ребус:



Малюнок 4

Як бачимо, у прикладі можуть використовуватися в повному обсязі не всі елементи, а тільки ті, які необхідні для алгоритму розв'язання. Причому послідовність блоків із діями ми визначаємо самостійно. І тільки на нас лежить відповіальність за коректність роботи такого алгоритму.

А тепер складніший приклад. Згадаймо стару головомку про Вовка, Козу і Капусту. Фермер мусить усе це перевезти на протилежний берег, однак усі четверо в човен не вмістяться. У човні є тільки два місця, одне з яких для фермера. І на березі не можна одночасно залишити козу з капустою або вовка з козою. Складемо алгоритм дій для фермера.

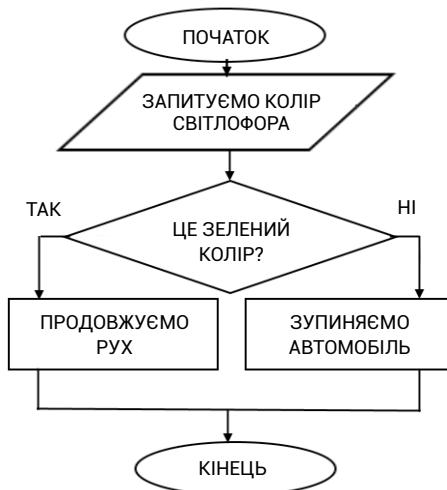
**Малюнок 5**

Як бачимо, ми маємо сформувати правильну послідовність дій. При виконанні кожного такого простого кроку потрібно відразу аналізувати, як він зможе в майбутньому вплинути на виконання всього алгоритму. Адже, якби ми спочатку перевезли, наприклад, капусту, а не козу, тоді б у фермера вже не було кози. А тепер розглянемо іншу ситуацію. Автомобіль рухається дорогою. Вдалині водій бачить світлофор, необхідно визначити дії водія залежно від кольору світлофора.

Спочатку з'ясовуємо, чи є вхідні дані для алгоритму? Так, є, це колір світлофора.

Далі визначаємо вихідні дані, тобто результат. Ця задача має два можливі варіанти правильної відповіді: продов-

ження руху або зупинка транспортного засобу. Тому її потрібно розв'язати за допомогою розгалуженого алгоритму з вибором дій. Після цього підбираємо послідовність наших дій відповідно до призначення блоків (мал. 6).



Малюнок 6

Такі алгоритми написати неважко, важливо дотримуватися основних правил побудови блок-схем і використовувати правильну послідовність дій.

А тепер час приступити до вивчення наступного способу опису алгоритму — програмного. Щоправда, спочатку потрібно ще розібратися з програмним забезпеченням.

4. Інсталяція Microsoft Visual Studio 2019

Microsoft Visual Studio — набір програмних продуктів від компанії Microsoft, які застосовуються для розробки рішень різного масштабу: від лабораторних робіт студентів до проектів корпоративного рівня. До складу Visual Studio входить інтегроване середовище розробки (IDE — *Integrated development environment*), численні інструментальні засоби й утиліти. За допомогою Visual Studio можна створювати як консольні додатки, так і додатки зі складним графічним інтерфейсом. Ми будемо використовувати продукти з лінійки Visual Studio в межах нашого процесу навчання.

Історія Visual Studio почалася в 1997 році, коли була випущена Visual Studio 97. Ви будете вивчати програмування з використанням Microsoft Visual Studio 2019. Це найактуальніша версія на сьогодні. Перед тим, як розглянути інсталяцію Visual Studio, розберемо питання редакції (*edition*) Visual Studio. Редакція Visual Studio — це версія Visual Studio, яка відрізняється від іншої редакції можливостями. Visual Studio 2019 має такі редакції: *Enterprise*, *Professional*, *Community*. *Enterprise* — це найповніша редакція Visual Studio.

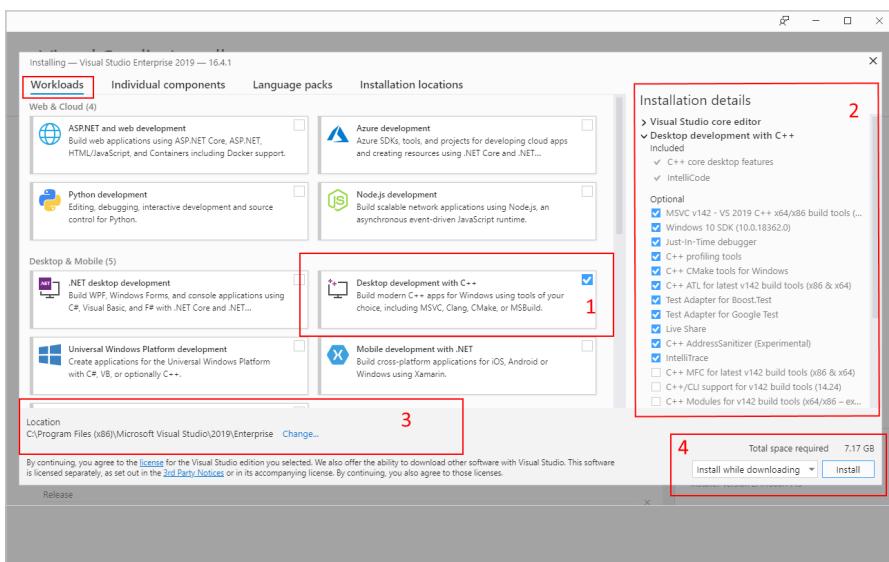
Кожна редакція має свою цінову політику. Винятком є *Community*. Це набір Visual Studio (ї), заточених під конкретні потреби. Наприклад, для розробки мобільних рішень або вебрішень, або рішень робочого столу тощо.

Використовувати Community можна абсолютно безкоштовно, зокрема і для комерційної розробки. Важливо зазначити, що різні редакції Visual Studio доступні на різних мовах.

Проте бажано вибирати оригінальну версію — англійську. Адже в межах вашої майбутньої професійної діяльності ви будете стикатися з колегами та проектами з інших країн, які напевно будуть використовувати англомовний інтерфейс.

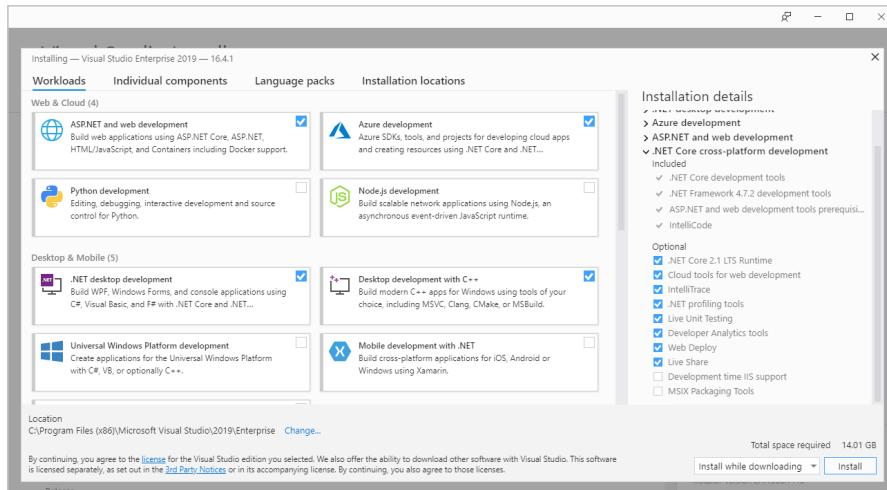
Ми розглянемо установку Visual Studio на прикладі Microsoft Visual Studio 2019 Enterprise.

Аби запустити інсталяцію Enterprise, змонтуйте ISO-образ за допомогою, наприклад, програми *Daemon Tools Lite* і запустіть файл *vs_setup.exe*, після чого перед вами з'явиться таке вікно (мал. 7):

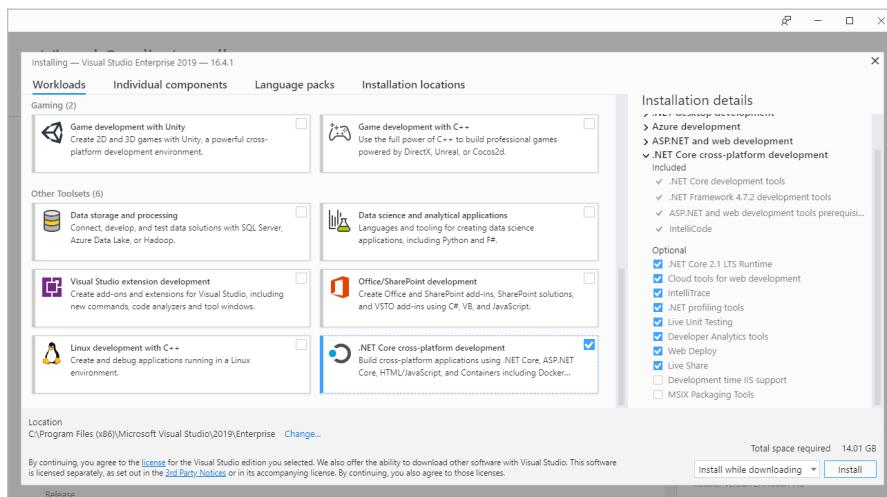


Малюнок 7

Урок № 1



Малюнок 7.1



Малюнок 7.2

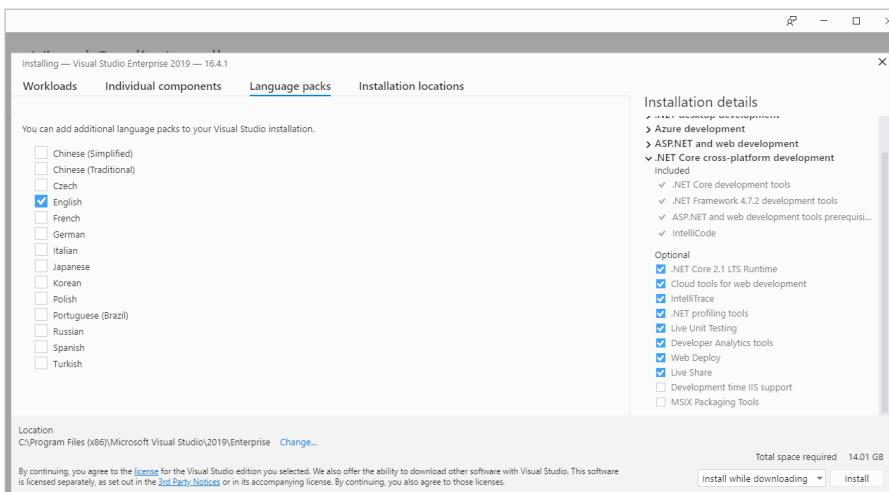
- У цьому вікні ви маєте виконати 4 основні пункти:
1. Вибір пакетів із засобами програмної розробки. Для всього навчання нам знадобляться:

- Розробка класичних додатків C++,
- Розробка класичних додатків .NET, ASP.NET і розробка вебдодатків,
- Розробка для Azure,
- Зберігання та обробка даних.

Будьте уважні, вибрали всі ці пакети, для установки вам знадобиться більше ніж 16 Гб, а також +20 Гб на диску з операційною системою для швидкої роботи середовища розробки в подальшому. Не турбуйтеся, для новачка досить вибрati пакет «Розробка» класичних додатків на C++, установка якого займає 10,6 Гб. І згодом ви навіть зможете просто довстановити відсутні компоненти, мовні пакети.

2. Вибір набору компонентів. Установник автоматично додає потрібні компоненти.

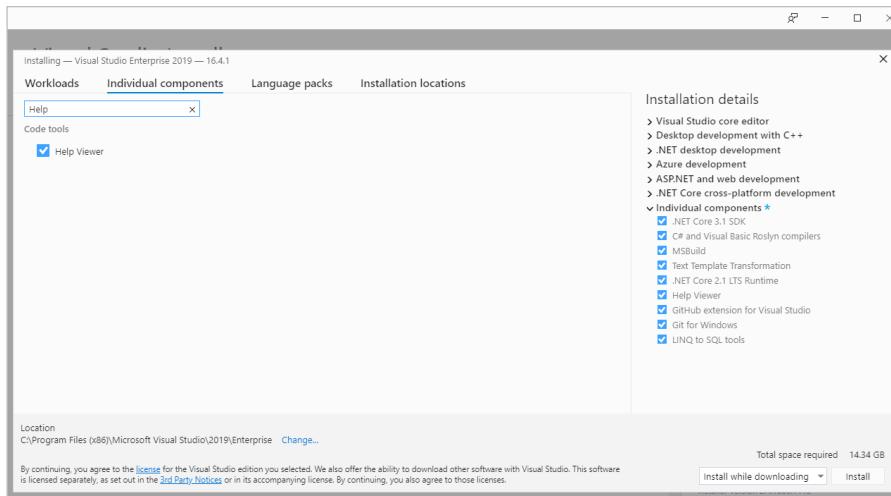
При навчанні та розробці програм вам дуже стане в нагоді довідкова інформація (MSDN) від Microsoft.



Малюнок 8

Тому її теж бажано додати в установку. Для цього переходимо на вкладку «Окремі компоненти» і ставимо галочку навпроти пункту «Вікно довідки» (мал. 8).

Мова інтерфейсу для Visual Studio визначається автоматично. Змінити її або додати нову можна у вкладці Мовні пакети. Бажано вибрati англійську мову. По-перше, це дозволить розширити ваш професiйний словниковий запас. По-друге, це позбавить вас вiд зaiвих несподiваних помилок при перенесеннi проекту на iншу машину. По-третe, найчастiше над великими проектами працює велика iнтернацiональна команда розробникiв, тому це є хорошим тоном (мал. 9).



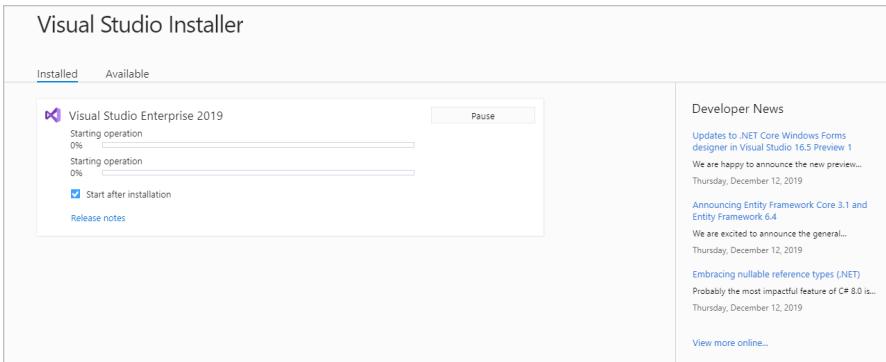
Малюнок 9

3. Визначаємо шлях для інсталяції Visual Studio.

4. Установка.

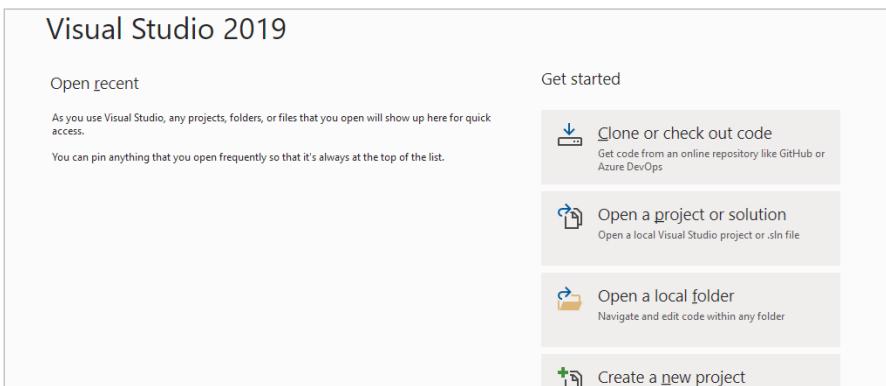
Натиснувши на кнопку «**Встановити**», ви пiдтверджуєте, що ознайомилися й погодилися з лiцензiйною угодою.

У вас має стартувати процес інсталяції. Він зайде деякий час, протягом якого ви зможете спостерігати за прогресом. Нижче ми наведемо приклад характерного для цього стану вікна (мал. 10).



Малюнок 10

Після закінчення встановлення, якщо все пройшло успішно, ви побачите таке вікно (мал. 11).

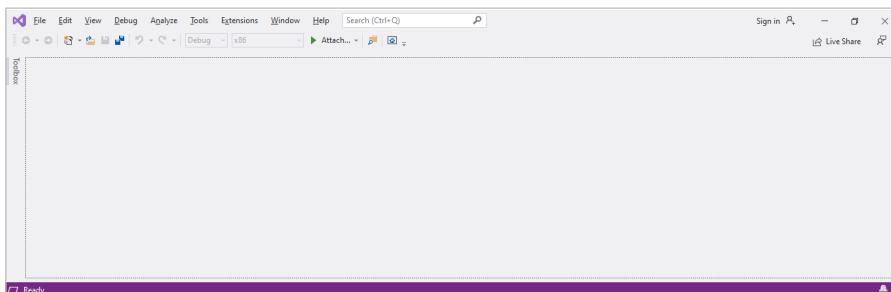


Малюнок 11

Натиснувши на нього, ви зможете запустити Visual Studio вперше, також ви можете використовувати звичний

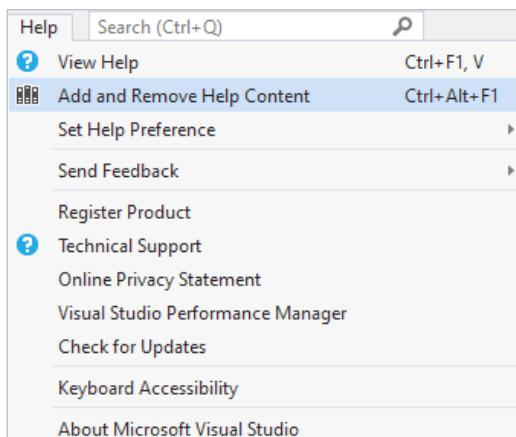
інтерфейс запуску програм Windows (меню «Пуск», ярлики тощо).

Запустимо Visual Studio, щоб переконатися в успішності наших попередніх дій. Під час первого запуску вас можуть попросити ввести ваш Microsoft Live ID. Проте, ви можете проігнорувати це і не вводити його (мал. 12).



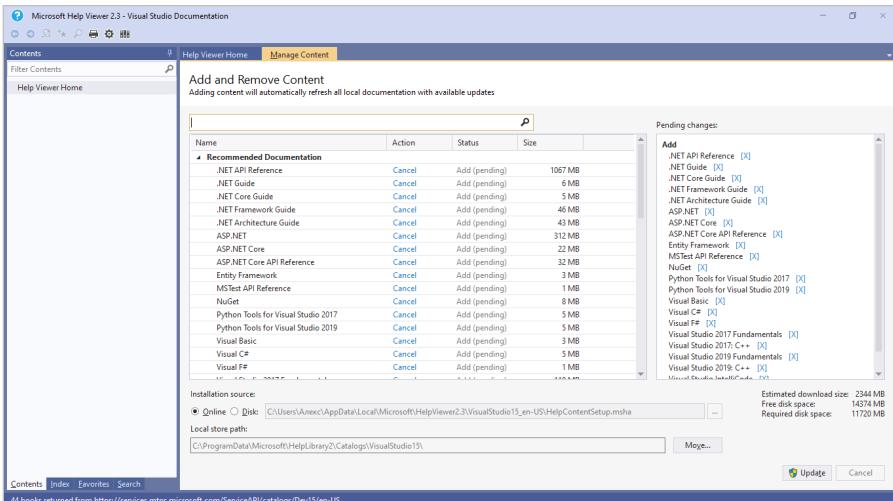
Малюнок 12

Аби встановити локальну копію довідкової інформації (MSDN) від Microsoft потрібно натиснути **Help -> Add and Remove Help Content** (мал. 13).



Малюнок 13

Ви побачите вікно з вибором довідкового матеріалу. Рекомендуємо всюди натиснути посилання «**Add**». Довідки багато не буває :) Після вибору матеріалів натисніть кнопку «**Update**» для їхньої локальної установки (мал. 14).



Малюнок 14

Якщо вам потрібно довстановити відсутні компоненти, потрібно просто запустити Visual Studio Installer. Під час установки Visual Studio 2019, він також закріпився в меню **Пуск**.

Процес інсталяції Microsoft Visual Studio Community 2019 дуже схожий на інсталяцію Enterprise, тому ми не будемо його дублювати.

От і все. Тепер ви можете переходити до наступних розділів уроку, де ми вчитимемося програмувати.

5. Перший проект

Один чоловік на ім'я Сергій описував свої враження про Прагу і написав мені: «Якщо ви потрапили на офіціанта, який не говорить українською, то беріть меню і показуйте пальцем, що ви хочете. Іноді схожі слова позначають різні речі. Наприклад, слово фрукти по-чеськи звучить майже як овочі. Вгадайте, що вам принесуть, якщо ви попросите овочевий салат? :)»

Невідомий вам Сергій хотів застерегти своїх друзів проти можливої помилки, але навряд чи він зміг би допомогти чим-небудь, якби замість офіціанта, нам довелося б спілкуватися з комп'ютером. Останньому, на жаль, невідомо, на який рядок меню ви тицяєте пальцем. Комп'ютер педантично виконує лише чіткі вказівки. І вказівки ці мають даватися за допомогою спеціальних команд. А команди складають цілі самостійні мови. Мови, зрозумілі комп'ютеру, — мови програмування.

Висновок: *щоб знайти спільну мову з офіціантом-чехом треба бути кмітливим. Але щоб знайти спільну мову з комп'ютером, треба цю мову знати.*

Вам вже відомо з першого розділу, що C++ — це мова програмування. Мова, яка дає нам можливість зрозуміло пояснити комп'ютеру, що ми від нього хочемо. Хоча, якщо бути до кінця відвертими, комп'ютер розуміє тільки одну мову — мову машинних кодів. Для прикладу, програма, яка виводить на екран фразу «Hello, world!», виглядає приблизно так на «рідній» для комп'ютера мові:

```

_ YH ; ў+. _ 51ЧN+f-H ; ў+ +f+ ¤д-ы+fO ; ¤+ЭмN¤¤¤+ | ФN¤¤¤- ;
- wGГ=xФN+MZP + @ A - | | - !+ L-!Tu . _ 51ЧN+f-
&ЛZФQW&бфЬN+&Л &ЛJf&OB_ФN+f; sЛ1ФN+Л&lt;
_t=ИФN+ GФu_ОФWfM+-чfЛN3+M-Ас°_> ] Л|Л+ЛМЛт _сбшьN+Л
9ШФN+тSЛSрФN+шN) fd! +RРбшьN+Л 9ШФN+тLСрФN+шC) + +Э
тАс°-щЗ-Л_ФN+f; s SЛФN+Л +u"бАФN+¤+МАс°ЎD At Э+- 3+
ыўы•ы°ыўыїLD$П$. _ 4ЧN++ . _ дL4 ў- uбSЛ дг4 + ε --fУf-
+ -+ЙA+@УN+Л- [Xf +tRPRVh•ў+ш_S Z+efO-fЛцf-d¤+S RfRfh
ш4 Л+Зы! SшC¤д-ы; ¤+ЬM-¤++¤_e fM+fO_fO ; f¤бЛЁdў
т3+ОшАс°Л$АФN+- dб4 ЙайS4 f_f _t дL4 ў- uSSЛ дг4 + ε
--fУf-+-+ЙA+@УN+Л- [ +eЙ] NfM+fO-fЛц+ыf-d¤+S RfRfh шз
л+Зы+Sш! $ РЕ L |7 p ¤! ' Р ОК Р ў+

```

Ви скажете, що в такому коді писати неможливо і будете абсолютно праві! Але так ми писати й не будемо. Для цього і потрібна мова програмування, щоб полегшити складання програм. Мови програмування поділяються на дві основні групи ІНТЕРПРЕТОВАНІ та КОМПІЛЬОВАНІ. Такий поділ пов'язаний із тим, яка спеціалізована програма переводить команди з мови програмування в машинну мову — КОМПІЛЯТОР або ІНТЕРПРЕТАТОР. Давайте з'ясуємо, в чому ж різниця між ними. Уявімо, що ми маємо файл, де міститься набір команд

Ситуація перша. Команди написані інтерпретованою мовою

Щоразу під час запуску програми інтерпретатор здійснює перевірку коду від рядка до рядка. Якщо помилок у синтаксисі немає, команди будуть перетворені на машинний код (набір інструкцій для процесора). Програма запуститься на виконання. Якщо є помилка,

інтерпретатор зупиниться і вам буде запропоновано її виправити і запустити програму знову. Але, навіть якщо помилок більше немає і програма остаточно дописана, при кожному її запуску буде спрацьовувати інтерпретатор і знову здійснювати перевірку коду. У такий спосіб можна зробити висновки, що машинна версія коду ніде не зберігається. Мінуси такого підходу полягають у тому, що швидкість запуску програми знижується, але відключити перевірку неможливо.

Ситуація друга. Команди написані компільованою мовою

Компілятор діє майже так само, як і інтерпретатор, тобто перевіряє код від рядка до рядка. Але якщо трапляється помилка, то він не зупиняється, а досліджує код до кінця, виявляючи всі наступні помилки і видаючи про них повідомлення. Крім того, компілятор формує спеціальний об'єктний файл із розширенням *.OBJ*. У цьому файлі зберігається текст програми, перекладений на машинну мову. Однак комп'ютер не працює безпосередньо з цим файлом.

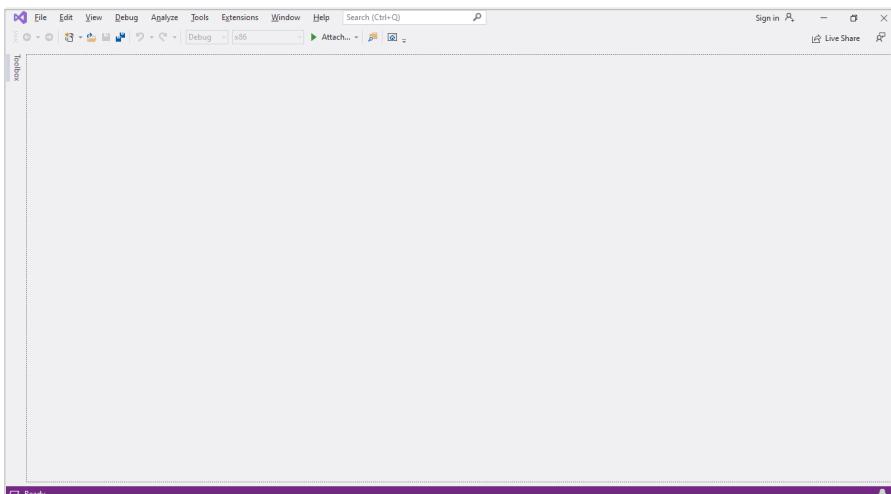
Є таке поняття, як компоновка або лінковка. Компонувальник — це ще одна спеціальна програма, яка збирає машинний код (з файлу з розширенням *.OBJ*) і різні допоміжні дані в єдиний виконуваний файл з розширенням *.EXE*. Такий файл може бути запущений на виконання як окрема, самостійна програма і в його запуску компілятор уже не бере участі.

Мова C++, вивчення якої ви розпочинаєте, є компільованою мовою. У нашому випадку, при роботі з оболонкою Microsoft Visual Studio, виклик компілятора

здійснюється автоматично і дозволяє перетворювати команди мови C++ у машинний код, що називається, «легким рухом руки».

Проба пера

Один із засновників мови С, Браян Керніган, сказав: «Єдиний спосіб вивчати нову мову програмування — писати на ній програми». Чим ми з вами зараз і займемось.

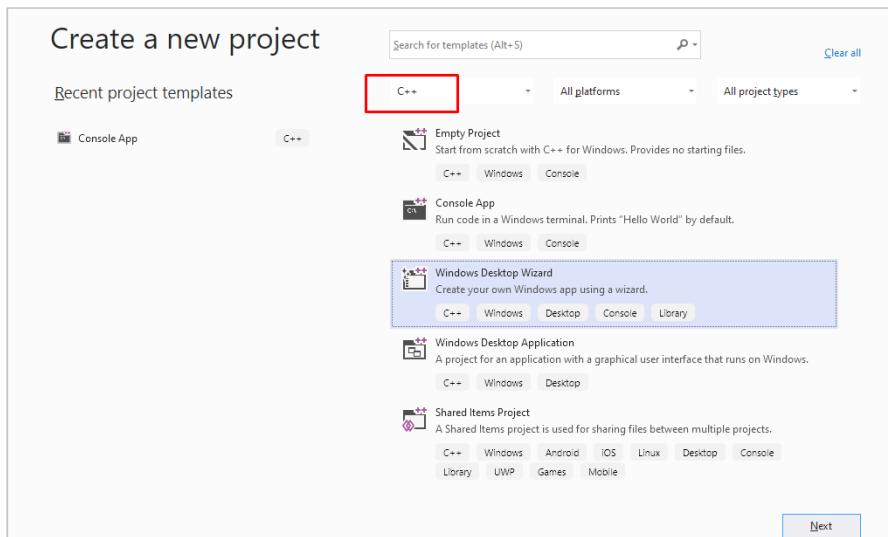


Малюнок 15

Так повелось у світі програмування, що перша програма на новій мові є програма «Hello world!». Аби написати свою першу програму, спочатку треба запустити ярлик програми Microsoft Visual Studio 2019 з пункту меню «Пуск» -> «Усі програми» -> «Microsoft Visual Studio 2019» або іншим звичним для вас чином. Після запуску ми побачимо зображення, подібне до того, що представлена на малюнку 15.

Зараз спробуємо створити проект, який і буде представляти нашу програму. Детальніше з тим, що таке проект, ми розберемося пізніше, коли будемо писати великих програм. Поки що будемо уявляти собі проект як об'єднання декількох файлів. А тепер давайте по кроках:

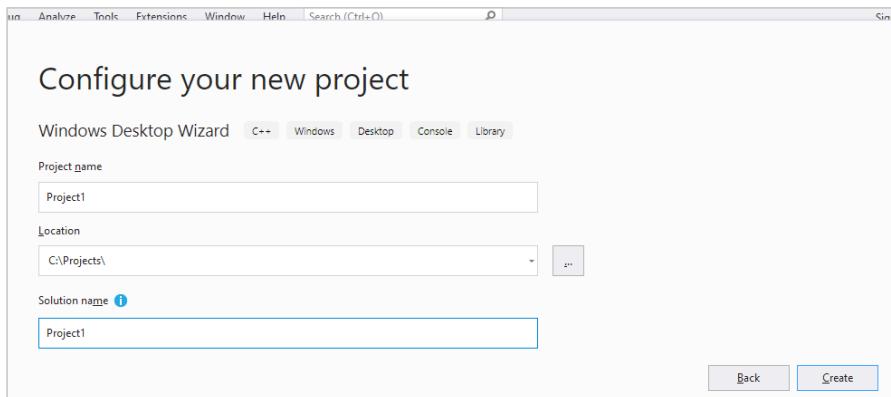
- Після того, як ви завантажили оболонку, виберіть пункт меню **File->New->Project**. Перед вами з'явиться діалогове вікно.
- У діалоговому вікні **Create New Project** (Новий проект) зі списку Мов виберіть C++ і **Windows Desktop Wizard** як тип проекту (мал. 16).



Малюнок 16

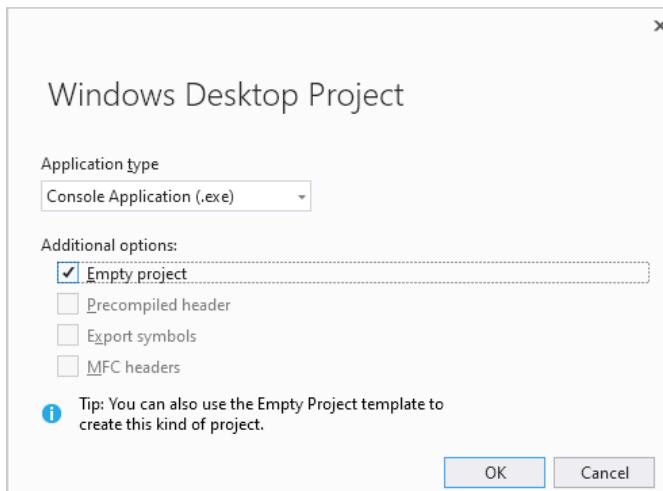
Після вибору натисніть кнопку **Next**.

Після цього в наступному вікні вам потрібно буде вказати ім'я проекту, шлях для збереження, ім'я рішення (вкажіть таке саме ім'я, як у проекті) (мал. 17)

**Малюнок 17**

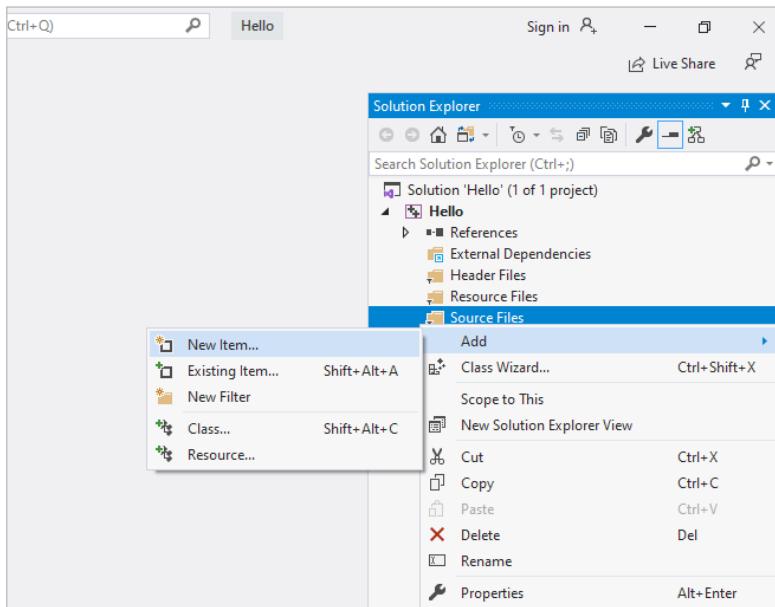
Після вибору натисніть кнопку **Create**.

Перед вами відкриється вікно налаштування властивостей проєкту — перевіряємо тип проєкту **Console Application**. Встановіть прапорець у полі **Empty Project** — це значить, що ми створюємо порожній проєкт. Тепер тисніть кнопку «**OK**» (мал. 18).

**Малюнок 18**

Отже, ми підготували місце для розташування нашої програми. Не зупиняємось на досягнутому — додамо в проект чистий файл. У ньому будемо набирати текст нашої програми. Для цього необхідно виконати наступні дії:

- Справа з'явилося віконце під назвою **Solution Explorer**. Для виклику цього вікна можна також використовувати поєднання клавіш **Ctrl+Alt+L**. У цьому вікні вам необхідно натиснути правою кнопкою на папці під назвою **Source Files**.
- У випадаючому меню вибираємо **Add->Add New Item...**

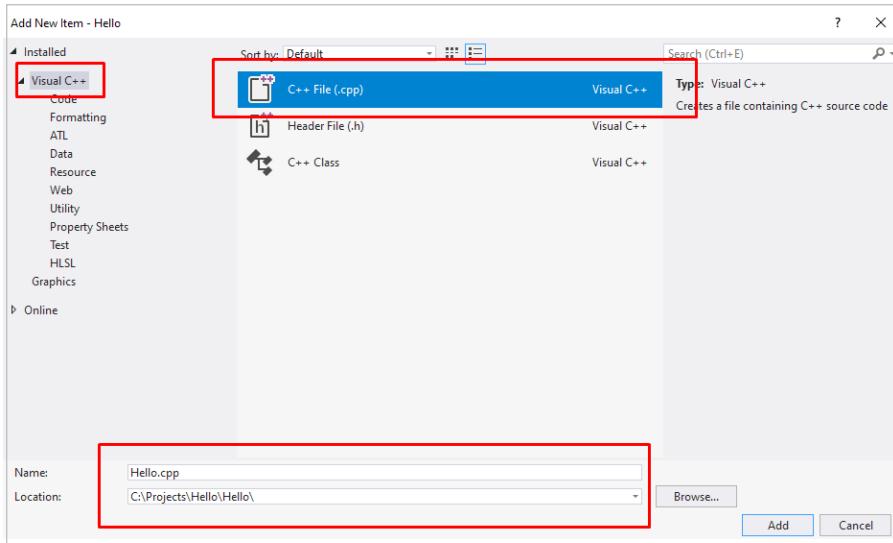


Малюнок 19

- Відкривається вікно вибору файлів. Знову маємо величезний вибір. Рекомендуємо зараз вибрати значок

C++ File (.cpp) (файл, який містить програму на мові C/C++).

- У текстовому полі **Name** (ім'я файлу) введіть ім'я файлу **Hello**. Натисніть кнопку **Add** (мал. 20).



Малюнок 20

Після натискання на кнопку **Add** у вас з'явиться текстова область, де ви зможете набрати свою першу програму.

Приклад першої програми на мові C++

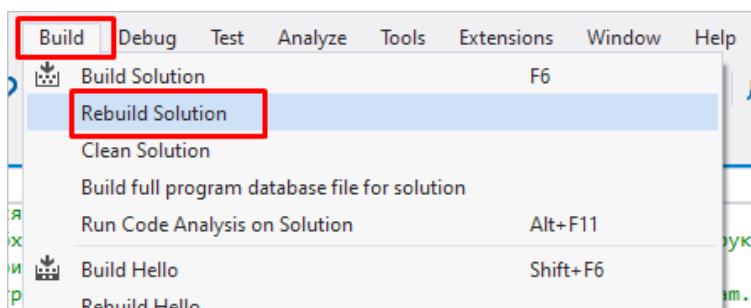
Перед тим як почати писати програму, давайте введемо для зручності поняття коментаря. Коментар — це замітки до програми, які призначені виключно для програміста. Компілятор ігнорує їх. Наприклад, за допомогою коментаря можна позначити, для чого використовується той чи інший рядок програми. У нижче описаному прикладі вказано, як правильно працювати з коментарями.

А тепер у текстовому вікні, що з'явилося, наберемо такий код:

```
// Це коментарі до програми
// Вони виділяються зеленим кольором
// Починаються коментарі двома рисками //
/* якщо необхідно створити багаторядковий коментар,
   використовується конструкція
*/
/* коментар */
/* Цей рядок підключає до програми бібліотеку
   під назвою iostream. Бібліотека – файл,
   у якому містяться описи різних функцій,
   що були реалізовані іншими програмістами.
   Ця програма отримала можливість використовувати
   функції, що розташовані в бібліотеці iostream
*/
#include <iostream>
/* У мові C++ є поняття простір імен.
   Цей простір визначає якусь область,
   на яку припадають дії оператора
   або функції. Аби використати оператор,
   що знаходиться в певному просторі, необхідно
   підключити цей простір у свою програму.
   Нижче підключається простір під назвою std
*/
using namespace std;
int main() // Початок програми, звідси програма
           // почне своє виконання
// Весь текст програми розташовується між
// фігурними дужками
{ // Це фігурна дужка
  // Наступний рядок виводить на екран
  // вітання Hello, World!
  // Ця дія виконується за допомогою cout<<
```

```
// Саме для цієї роботи підключалася бібліотека
// і простір імен, у яких він розташовується
cout<<"Hello, World!\n";
// У кінці команди стоїть крапка з комою.
// Цим знаком МУСИТЬ закінчуватися
// кожна команда в мові C++.
// повертаємо 0 операційній системі
// це означає, що програма завершилася без помилок
return 0;
}
```

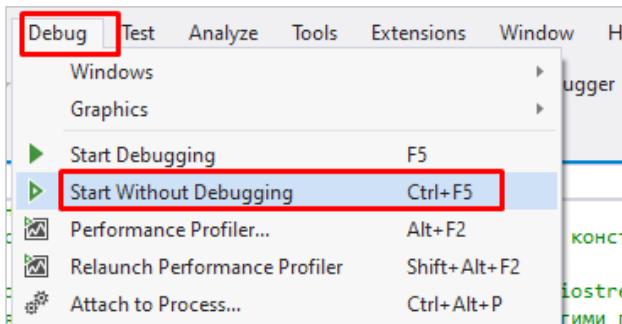
Як ви вже знаєте, комп'ютер розуміє тільки мову машинних кодів. І перш ніж програма буде виконуватися комп'ютером, потрібно її перевести в мову машинних кодів. Ви пам'ятаєте, що це будемо робити не ми, а КОМПІЛЯТОР. Проекти, написані на C++, включають у себе багато файлів. Прокомпілюємо відразу всі файли, включені в проект. Для цього в рядку меню виберіть пункт меню **Build** (*побудувати*), потім **Rebuild Solution** (*перебудувати все*).



Малюнок 21

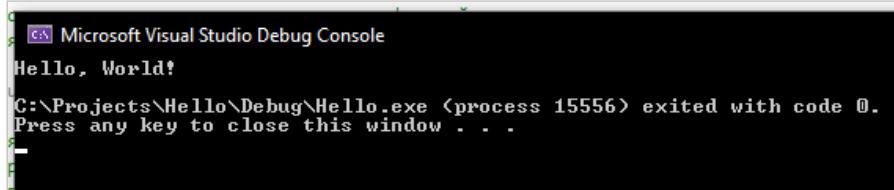
Сподіваємося, ви набрали текст без помилок. Наша програма успішно переведена на мову машинних кодів і

її можна запустити на виконання. Запустити програму на виконання просто. У меню **Debug** вибрать **Start Without Debugging** (мал. 22).



Малюнок 22

Програма відпрацює і ви побачите таке от вікно (мал. 23):

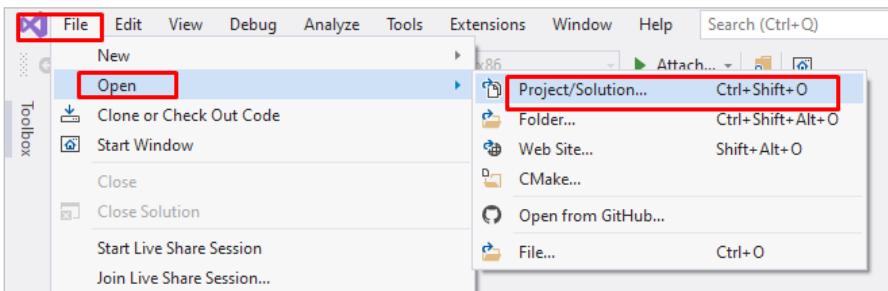


Малюнок 23

Вітаю! Було досить легко. Правда? Аби закінчiti роботу програми, натисніть будь-яку клавішу на клавіатурі (*Press any key to continue*).

Відкриття збереженого проекту

Щоб відновити раніше збережений проект на диску, запустіть Visual Studio (якщо вона у вас ще не завантажена). Виберіть у меню **File** пункт **Open -> Project/Solution** і вкажіть ім'я вашого проекту (мал. 24, 25).

**Малюнок 24**

Після того як проект було відкрито, ви можете продовжити роботу над ним.

6. Вивід даних

Ви вже знаєте, що за допомогою команди **cout<<** ми можемо виводити на екран різні текстові рядки. Однак, щоб компілятор зрозумів таку команду, ви маєте пам'ятати про наступні три основні моменти:

1. У заголовку програми має бути присутній рядок

```
#include <iostream>
```

2. Перед використанням команди необхідно підключити простір імен, до якого належить команда **cout**.

```
using namespace std;
```

3. Рядок, який ми хочемо вивести на екран, використовуючи **cout <<**, ми обов'язково записуємо в лапки. Наприклад:

```
cout<< // тут пишемо те, що хочемо";
```

Команда **cout<<** не тільки виводить на екран рядки, але й дозволяє їх оформляти. Для оформлення виводу рядка використовують спеціальні керівні символи, що являють собою комбінацію символу «\» і символу, що визначає дію, яку необхідно провести над рядком. Ці керівні символи називаються Escape-послідовностями. Нижче наводяться деякі з них:

```
\b // Видалення останнього виведеного символу  
\n // Перейти на початок нового рядка
```

```
\t // Перейти до наступної позиції табуляції
\\ // Вивести зворотну рису \
\" // Вивести прямі подвійні лапки "
\' // Вивести одинарні лапки '
```

Існування останніх трьох Escape-послідовностей спочатку завжди викликає легке здивування. Навіщо користуватися керівними символами, якщо можна просто написати: «**а**бо \ або ? Відповідь лежить на поверхні — усі ці три символи є операторами і якщо їх «просто написати», то компілятор сприйматиме їх як оператори. Наприклад, коли слово використовується в переносному сенсі, воно пишеться в лапках. Припустимо, вам потрібно вивести такий текст на екран:

```
The Man in red was "old friend" of John...
```

Якщо ви не використовуєте Escape-послідовності, то очевидно, що ваша команда виглядатиме так:

```
cout<<"The Man in red was "old friend" of John...";
```

І це призведе до неминучої помилки. Компілятор сприйме тільки частину рядка, а саме **cout<<"The Man in red was"**. Подвійні лапки після «was» він вважатиме закриваючими, а все інше сприйме як невірний синтаксис мови. Така програма, природно, не запуститься на виконання. Правильний варіант такий:

```
cout<<"The Man in red was \"old friend\" of John...";
```

Тепер давайте поговоримо про те, де саме в **cout<<** можна вказувати Escape-послідовності. Найголовніше,

що вам необхідно знати — Escape-послідовність завжди має бути всередині лапок, тому що це текст, а далі ваші можливості практично не обмежені. Наприклад, так:

```
cout<<'" My name is'"<<'" - Ira\n "';  
cout<<'"I'm from Odessa\n "';  
cout<<'"My eyes are blue"'<<"\n "'<<"That's all!!!!";
```

У результаті роботи цієї команди ми побачимо на екрані:

```
My name is - Ira.  
I'm from Odessa  
My eyes are blue  
That's all!!!!
```

Практичний приклад використання cout<<

Напишемо програму, яка виводить на екран коротку довідку про вивчені нами Escape-послідовності. Ось що ми хочемо побачити на екрані:

```
\b' // Backspace  
\n' // New line  
\t' // Horizontal tab  
\\" // Backslash \  
\\" // Double quotation mark "  
\'' // Single quotation mark '
```

Запускаємо середовище *Visual Studio 2019*. Створюємо новий проект під ім'ям **EscapeSequences**. Набираємо код, який розташовується нижче.

```
// Заголовок
#include <iostream>
// визначення простору імен, де є cout<<
using namespace std;

// Головна функція
int main()
{
/* Наступна команда через 4 табуляції виводить текст
   Escape Sequences і переводить вивід на наступний
   рядок
*/
    cout<<"\t\t\tEscape Sequences\n";
    // Виводить порожній рядок
    cout<<"\n";
/* Через 2 табуляції виводить текст \b, і ще через
   1 табуляцію Backspace. Потім \n переводить вивід на
   наступний рядок
*/
    cout<<"\t\t\b" << "\tBackspace\n";
    // Виводить порожній рядок
    cout<<"\n";
/* Через 2 табуляції виводить текст \n, і ще через
   1 табуляцію New line
   Потім \n переводить вивід на наступний рядок
*/
    cout<<"\t\t\n" << "\tNew line\n";
    // Виводить порожній рядок
    cout<<"\n";
/* Через 2 табуляції виводить текст \t, і ще через
   1 табуляцію Horizontal tab. Потім \n переводить
   вивід на наступний рядок
*/
    cout<<"\t\t\t" << "\tHorizontal tab\n";
    // Виводить порожній рядок
    cout<<"\n";
```

```

/* Через 2 табуляції виводить текст \\, і ще через
   1 табуляцію Backslash \
   Потім \n переводить вивід на наступний рядок
*/
cout<<"\t\t\\\\\"<<"\tBackslash \\\n";
// Виводить порожній рядок
cout<<"\n";
/* Через 2 табуляції виводить текст \t,
   і ще через 1 табуляцію Double quotation mark "
   Потім \n переводить вивід на наступний рядок
*/
cout<<"\t\t\"<<"\tDouble quotation mark \"\n";
// Виводить порожній рядок
cout<<"\n";
/* Через 2 табуляції виводить текст \',
   і ще через 1 табуляцію Single quotation mark '
   Потім \n переводить вивід на наступний рядок
*/
cout<<"\t\t\'<<"\tSingle quotation mark \'\\n";
// Виводить порожній рядок
cout<<"\n";
return 0;
}

```

Відкомпілюємо програму (**Build -> Rebuild Solution**). Якщо виникло багато помилок, то згадайте такі правила:

- Якщо в програмі будуть виводитися повідомлення на екран, то в початок програми записується рядок **#include<iostream>** і підключається простір імен, до якого належить команда **cout (using namespace std;)**.
- Кожна програма має містити функцію з ім'ям **main()**. Робота програми починається з виконання цієї функції.
- Команди функції **main()** знаходяться всередині фігурних дужок **{ }**.

- Усі команди обов'язково мають закінчуватися символом крапка з комою.

І запустимо її (**Debug -> Start Without Debugging**).

P. S.

Ви, мабуть, звернули увагу на те, що ми вживаємо лише латинські символи при виводі даних на екран. Справа в тому, що кожен символ має в будь-якій операційній системі свій числовий код. І система ідентифікує його саме за цим кодом. Символьні коди кирилиці в консольному додатку під управлінням Windows не збігаються, тому програма з використанням кирилиці буде працювати некоректно. Наприклад, якщо ми напишемо в Windows:

```
cout << "Утро";
```

на екран виведеться:

```
µ€Ёю
```

Це легко пояснити тим, що у Windows, наприклад, літера «о» — 238, а в консолі цьому коду відповідає літера «ю». А коди латиниці співпадають. Надалі ми з вами навчимося виправляти цю ситуацію.

У стандарті мови C++ 11 програмістам дали нову можливість для виводу спеціальних символів на екран. Для цього необхідно використати рядки «raw». Позначення рядка як «raw» дає вказівку компілятору трактувати його посимвольно. Аби позначити рядок таким чином, потрібно використати такий формат.

```
R "(текст_рядка)"
```

R вказує на те, що це «raw»-рядок. Вміст рядка обов'язково виділяти дужками.

Наведемо кілька прикладів:

```
cout<<R"(hello\nworld)"; // на екрані hello \ nworld
cout<<R"("Test 'string'\t")"; // на екрані "Test
                                // 'string'\t"
cout<<R"((Such brackets))"; // на екрані
                                // (Such brackets)
```

7. Типи даних

Після того, як ви прочитали попередні розділи уроку, вам вже буде не складно написати програму, яка виводить що-небудь на екран.

```
// Заголовок
#include <iostream>
// визначення простору імен, де є cout<<
using namespace std;
// Головна функція
int main()
{
    // вивід фрази "Вітаємо з добрим початком!!! :)"
    // фраза виводиться через три табуляції,
    // потім додаються два порожні рядки
    cout<<"\t\t\tCongratulation with"
        "good beginning!!! :)\n\n";
    return 0;
}
```

Ось, власне, і все, що ви вмієте (поки).

Людина ніколи не повинна зупинятися на досягнутому. Власне, вас обов'язково має зацікавити не тільки те, як виводити дані на екран, але й як оперувати цими даними. Наприклад, проводити будь-які обчислення. Безруких жонглерів не буває, і, поки частина куль знаходиться в повітрі, решту циркач тримає в руках. Аби що-небудь зберігати (зокрема, дані), необхідно мати сховище. Для нашої програми таким сховищем буде оперативна пам'ять. Перш ніж десь щось розмістити, необхідно підібрати відповідну упаковку. Скажімо, ви навряд чи станете наливати

молоко в сірникову коробку. У програмуванні, перед тим як розмістити інформацію в оперативній пам'яті, ви обов'язково маєте визначити характер цієї інформації. Отже, **типи даних**.

Тип даних — поняття, що визначає максимальний розмір (у байтах) і тип інформації, яка буде використовуватися програмою.

Програмування частково відображає об'єкти зовнішнього світу, неабияк їх спрощуючи. На початку вивчення ми зіткнемося з речами, з якими, по суті, ви стикалися багато разів. Давайте умовно розділимо всі типи даних на наступні групи:

1. Числові.
2. Символьні.
3. Логічні.

Далі ми розглянемо низку ключових слів, які використовуються в мові C++ для позначення типів даних.

Числові типи

Числа, як відомо, бувають цілі й дійсні. Дійсні числа ми будемо називати числами з рухомою точкою. Особливо зазначимо, що кома, що відокремлює цілу частину від дробової, змінюється на точку. Наприклад, 7,8 у C++ записується як 7.8.

Змінні, у яких ми будемо зберігати значення дійсних чисел, будуть оголошуватися типом **float** або **double**. У чому різниця між цими типами? Тип **float** описує числа з рухомою точкою одинарної точності, а **double** — подвійної. Пояснимо, що в математиці точність визначається

кількістю знаків після коми. Подвійною точністю називають метод представлення чисел із подвоєною, порівняно зі звичайною, кількістю цифр. Ось характеристики типів для чисел з рухомою точкою:

Таблиця 1

Пояснення	Тип	Розмір у байтах
Описує дійсні числа одинарної точності	float	4
Описує дійсні числа подвійної точності	double	8

Окрім дійсних, у C++ передбачено три типи позначення цілочисельних даних. У таблиці наведені основні характеристики цих типів:

Таблиця 2

Пояснення	Тип	Розмір у байтах	Діапазон значень
Описує цілі числа	int	4	від -2147483648 до 2147483647
Описує короткі цілі числа	short	2	від -32768 до 32767
Описує довгі цілі числа	long	4	від -2147483648 до 2147483647
Описує довгі цілі числа	long long	8	від -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807

Символьний тип

Тип призначений для зберігання тільки одного символу.

Таблиця 3

Пояснення	Тип	Розмір у байтах
Описує символи	char	1

Логічний тип

Тип призначений для зберігання логічних даних. Детальніше ми познайомимося з ним пізніше. Логічні дані можуть приймати одне з двох значень: істина (**True**) або брехня (**False**).

Таблиця 4

Пояснення	Тип	Розмір у байтах	Значення
Описує логічні значення	bool	1	true false

Примітка. Якщо необхідно виключити з діапазону типу даних від'ємні значення, перед назвою типу потрібно вказати ключове слово *unsigned*. Наприклад, *unsigned int*. Такий тип включатиме в себе тільки додатні значення від 0 до 4294967294.

Отже, ми з'ясували, які бувають типи даних і які ключові слова мови C++ використовуються для їхнього позначення. У висновку треба зазначити, що мова C++ є чутливою до регістру (тобто ВЕЛИКІ і малі літери в ньому це не одне й те саме). Зверніть увагу на те, що всі вище описані типи даних записані малими літерами. Слідкуйте за цим, тому що **int** — це тип даних, а **INT** — помилка.

У наступній темі ми розглянемо застосування типів даних на практиці.

8. Змінні константи

Ми з вами вже познайомилися з типами даних і знаємо, як класифікується інформація для зберігання. Залишилося з'ясувати, як дані записати в оперативну пам'ять і як до них потім звертатися, щоб використати або змінити.

Отже, мінливі дані домовимося називати **ЗМІННИМИ**, а постійні дані — **КОНСТАНТАМИ**.

Змінна — область оперативної пам'яті, що володіє власним ім'ям і призначена для зберігання даних, які можуть бути змінені.

Константа — область оперативної пам'яті, що володіє власним ім'ям і призначена для зберігання постійних даних.

Ось приклад констант: усім відома кількість днів у тижні та кількість місяців у році... Вона не змінюється за жодних обставин, — тому ці значення — константи.

А ось наш вік — величина змінна. Сьогодні комусь 26 років, а за рік буде 27.

З визначень стає зрозуміло, що для пошуку даних у пам'яті їм дають імена (за аналогією з тим, що на речі в багажному вагоні вішають бірочки). У програмуванні їх називають ідентифікаторами. Одна з перших проблем, яку вирішують батьки новонародженого — це вибір імені для нього. Чи накладає ім'я відбиток на характер людини, на його долю — питання складне й суперечливе. Можна почути абсолютно протилежні думки на цей рахунок. Але той факт, що ім'я (ідентифікатор), що дается новій змінній (константі), має бути інтуїтивно зрозуміле

і пояснювати призначення змінної не буде оскаржувати жоден більш-менш досвідчений програміст.

Імена даним даються, дотримуючись певних правил. Ці правила не можна порушувати!

Правила складання імен

В імені допустимо використовувати тільки наступні символи:

1. **Великі та малі літери латинського алфавіту.** При цьому не забувайте про чутливі до регістру мови. Наприклад, **Age** та **age** — це два різні імені.
2. **Цифри.** Однак цифра не може бути використана як перший символ. Тобто **Name1** є допустимим, а **1Name** — ні.
3. **Символ підкреслення «_».** Справа в тому, що ви маєте пам'ятати, що пробіл теж є символом, і цей символ неприпустимий в імені змінної. Його замінить знак підкреслення, який поліпшить виразність імен. Наприклад, порівняйте: **ageofman** та **Age_Of_Man**.

При визначенні імені для змінної запам'ятайте наступне:

4. **Не можна називати змінну ключовими словами мови програмування.** Ключове слово — слово, зарезервоване під синтаксис мови програмування (**int**, **float**, **double** тощо). У Visual Studio ключові слова підсвічуються синім кольором, це як мінімум призведе до плутанини.
5. **Небажане існування двох ідентифікаторів з однаковими іменами.**
6. **Не можна використовувати жодні інші символи, окрім допустимих** (див. вище).

Оголошення та використання змінних і констант

Тепер ми володіємо всією інформацією для створення (оголошення) змінної. Залишилося лише з'ясувати, який загальний синтаксис:

1. **Тип_даних ім'я_змінної;** — у цьому випадку в оперативній пам'яті буде виділений осередок розміром, що відповідає заданому типу. І в цьому осередку буде присвоєно обране вами ім'я. Що ж там буде міститися? У щойно створену змінну буде записане випадкове число, яке визначається операційною системою. Це число буде міститися в пам'яті доти, доки ви не заповните змінну іншим значенням за допомогою спеціального оператора присвоєння **=**.
2. **Тип_даних ім'я_змінної = значення;** — існує й така можливість — заповнити змінну значенням прямо при створенні. Такий процес ми будемо називати ініціалізацією.
3. **Const тип_даних ім'я_змінної=значення;** — а це оголошення константи. Основні моменти полягають у тому, що незалежно від типу даних перед ним вказується ключове слово **const**. Okрім того, константа обов'язково має бути ініціалізована при створенні. Поміняти її значення згодом буде неможливо.

Вивід значення змінної на екран

Вивід значення змінної на екран здійснюється за допомогою **cout<<**

```
cout<<ім'я_змінної; // лапки в цьому випадку не вказують
```

Можна показувати вміст декількох змінних через <<

```
cout<<iм'я_змінної1<<iм'я_змінної2;  
// лапки в цьому випадку не вказують
```

Можна чергувати показ вмісту змінних з текстовими повідомленнями та Escape-послідовностями через <<cout<<"Текст"<<iм'я_змінної1

```
<< "Текст"<<iм'я_змінної2<<"\n";
```

Показ вмісту констант здійснюється повністю аналогічно зі змінними.

Практичні приклади

Наведемо кілька прикладів створення й ініціалізації змінних і констант для різних типів даних.

Ціличисельні змінні та константи

З цілими числами ми стикаємося повсюдно: вік, кількість стільців, кількість кімнат, кількість днів у тижні тощо.

Змінні, у яких будуть зберігатися цілі числа, ОГОЛОШУЮТЬСЯ так:

```
int Age;
```

Про що говорить цей рядок? Що в змінній на ім'я **Age** (вік) буде зберігатися ціле значення. Слово **int** оголошує ТИП значення змінної на ім'я **Age**.

Тепер, наприклад, ми хочемо внести в змінну **Age** значення **34**. Як це зробити?

```
Age = 34;
```

Цей рядок читається так: «Змінній **Age** присвоїти значення **34**». Ще раз подивимося на оператор присвоєння: **Age = 34**. Зліва від знака рівності стоїть ім'я змінної, якій присвоюється значення. А справа стоїть те значення, яке присвоюється.

Константа, у якій буде зберігатися ціле число оголошується так:

```
const int Count_Days_in_Week = 7;
```

Про що говорить цей рядок? Слово **const** підкреслює, що оголошується константа. Слово **int** повідомляє, що константа буде цілим числом. Потім йде ім'я константи **Count_Days_in_Week** та її значення **7**.

Тепер розберемо, як обчислювати значення змінної. Для чого це потрібно? Простий приклад: як порахувати, скільки годин у 2000 році? Невже ви хочете порахувати це число самі?

Насправді, досить легко змусити комп'ютер це зробити самостійно. Нам лише потрібно написати формулу цього обчислення.

У 2000 році 366 діб, у добі 24 години. Отже, формула розрахунку кількості годин у 2000 року така: 366 помножити на 24.

У мові C++ як знак множення використовують * (зірочка, комбінація **Shift+8**).

Розробимо програму, яка рахує скільки ж годин у 2000 році. Перед створенням програми рекомендується коротко накидати її алгоритм.

Алгоритм — послідовність дій, спрямована на розв'язання поставленої задачі.

Дано: кількість днів у році — 366. Це значення не змінюватиметься, тому оголосимо його константою цілого типу на ім'я **DayIn_2000Year**. Кількість годин у добі — 24. Теж не змінюється. Оголосимо її константою цілого типу на ім'я **HourInDay**. У нашій програмі буде єдина змінна, у неї ми запишемо результат розрахунку. Назовемо цю змінну **HourIn_Year2000**. Вона буде цілого типу (**int**).

Алгоритм буде наступний:

1. Оголошення та ініціалізація змінних і констант.
2. Підрахунок результату.
3. Вивід результату на екран.

Імена змінних ви можете придумати самі (не забувайте тільки про правила складання імен змінних).

А тепер, як завжди, створимо новий проект і введемо такий код:

```
// Заголовок
#include <iostream>
//визначення простору імен, у якому є cout<<
using namespace std;
// Головна функція
int main()
{
    // вивід порожнього рядка
    cout<<"\n";
    // Оголошуємо цілочисельні константи
    int DayIn_2000Year=366;
    int HourInDay=24;
    // оголошуємо цілочисельну змінну
    int HourIn_Year2000;
    // обчислюємо шукане значення та
    // поміщаємо його в змінну HourIn_Year2000
    HourIn_Year2000=DayIn_2000Year*HourInDay;
```

```

    // виводимо значення змінної
    // HourIn_Year2000 на екран
    cout<<"\t\t In 2000 year "<< HourIn_Year2000;
    cout<<" hours\n ";
    return 0;
}

```

Усе! Компілюйте програму!

Дійсні змінні та константи

Приклад оголошення та ініціалізації:

```

float Weight;
Weight=12.3452;
double weight_atom;
weight_atom= 2.5194e+017;

```

Що означає число $2.5194e+017$?

Це короткий запис дійсних чисел. Називається він експоненційною формою запису чисел. Повідомляємо вам секрет розшифровки написаного. Цим набором символів описується число 25194000000000000000 або $2,1594 \times 10^{17}$.

- $3.4E-008$ розшифровується так: $3,4 \times 10^{-8}$, що аналогічно $3,4:10^8$.
- $-1.5E+003$ розшифровується як $-1,5 \times 10^3$.

Числа з рухомою точкою типу **float** можуть змінюватися від $-3,4 \times 10^{-38}$ до $3,4 \times 10^{38}$.

Значення від $-3,4 \times 10^{-38}$ до $3,4 \times 10^{-38}$ вважаються рівними нулю.

А тепер давайте попрацюємо з дійсними числами на практиці.

Напишемо програму, яка буде розраховувати вартість покупки. Нехай програма використовує ціну товару (*Cost*), кількість купленого товару (*Count*), і, з огляду на знижку (*Discount*), обчислює вартість покупки (*Price*).

Створимо новий проект *Pokupka* і введемо текст такої програми

```
// Заголовок
#include <iostream>
// визначення простору імен, де є cout<<
using namespace std;
// Головна функція
int main()
{
    // Оголошуємо змінну Discount
    float Discount=0.05;
    // Оголошуємо змінну
    Cost float Cost=10.50;
    // Оголошуємо змінну
    Count int Count=5;
    // Оголошуємо змінну
    Price float Price;
    // Обчислюємо значення змінної Price
    Price=Count*Cost-Count*Cost*Discount;
    // Виводимо підсумкову вартість товару зі знижкою
    cout<<"Please, pay:"<<Price<<"\n";

    return 0;
}
```

Компілюйте програму і відправляйте її на виконання. Те, що ви маєте побачити на екрані, наведено нижче (мал. 26).

Символьні та логічні змінні і константи

У цьому уроці ми не будемо наводити приклади використання символьних і логічних змінних констант. Їхнє призначення більш детально буде описано в майбутньому. Обговоримо лише оголошення та ініціалізацію.

```
// Логічна змінна
bool Flag;
Flag=true;
// Один символ завжди вказується в одинарних лапках
char Symbol='A';
/* Escape-послідовність розглядається
компілятором як один символ і, відповідно,
може бути записана в змінну або константу
типу char
*/
const char NewLine='\n';
cout << NewLine // показує порожній рядок
```

9. Ввід даних

Ви вже знайомі з операцією виводу інформації на екран комп'ютера — **cout**, але в більшості програм потрібно не тільки виводити будь-яку інформацію на екран, але й мати можливість ввести в комп'ютер будь-які дані з клавіатури. У попередньому розділі була наведена програма розрахунку знижки. Природно, що такі параметри, як ціна та кількість товару, було б непогано ввести з клавіатури на етапі виконання програми. Давайте розглянемо, як ви це можете зробити. Якщо нам потрібно ввести дані в комп'ютер, то будемо користуватися командою **cin**. Як нею користуватися?

Синтаксис оператора введення:

```
cin>>им'я_змінної;
```

Ім'я_змінної вказує на змінну, у яку потрібно помістити дані, введені з клавіатури:

```
cin>>Age;
```

Ця команда поміщає число, введене з клавіатури, у змінну з ім'ям **Age**. Аби ввести число в змінну **Number**, потрібно лише набрати ось таку команду:

```
cin>>Number;
```

Введення відразу декількох змінних записують так:

```
cin>>им'я_змінної1>>им'я_змінної2>>...>>им'я_змінноїN;
```

Список імен змінних має містити імена всіх змінних, у які ви хочете ввести дані з клавіатури. Список імен може

складатися з будь-якої кількості імен змінних, розділених комбінацією символів **>>**.

```
cin>>Quantity>>Price>>Discount;
```

Давайте додамо в програму *Pokirka* введення даних з клавіатури:

```
// Заголовок
#include <iostream>
// визначення простору імен, де є cout<<
using namespace std;
// Головна функція
int main()
{
    // Оголошуємо змінну Discount
    float Discount=0.05;
    // Оголошуємо змінну Cost
    float Cost=10.50;
    // Запрошення ввести ціну товару
    cout<<"What's the cost?\n";
    // Введення значення в змінну Cost
    cin>>Cost;
    // Оголошуємо змінну Count
    int Count=5;
    // Запрошення ввести кількість
    cout<<"How much?\n";
    // Введення значення в змінну Count
    cin>>Count;
    // Оголошуємо змінну Price
    float Price;
    // Обчислюємо значення змінної Price
    Price=Count*Cost-Count*Cost*Discount;
    // Виводимо підсумкову вартість товару зі знижкою
    cout<<"Please, pay:"<<Price<<"\n";
    return 0;
}
```

Тепер ви побачили особливість роботи оператора `cin >>`. Щойно програма зустрічає цей оператор, вона зупиняється і чекає на реакцію користувача — доти, доки користувач не введе дані і не натисне «Введення» (`Enter`). Тільки після цього продовжиться виконання.

На прикладі ще раз попробуємо з вводом і виводом. Напишемо програму-обманщика: програма пропонує пограти в числа, хто загадає більше число, той і виграє.

Створимо новий проект *Game* і введемо такий текст:

```
// Заголовок
#include <iostream>
// визначення простору імен, де є cout<<
using namespace std;
// Головна функція
int main()
{
    // Запрошення "Давай грать!"
    cout<<"Let's play!\n";
    // Оголошення змінної і
    int i;
    // Запрошення "Введіть число"
    cout<<"Enter a number:";
    // Введення числа
    cin>>i;
    // Вивід числа, яке "загадав" комп'ютер
    cout<<"I have "<<i+1<<"\n";
    // Вивід результату гри
    cout<<"I'm winner!\n";
    return 0;
}
```

Відкомпілюйте програму. Працювати з нашою програмою легко. Просто введіть будь-яке число; але постійно виявляється, що в комп'ютера число більше, і він виграє.

Ось, що ви побачите на екрані при запуску програми, якщо на запит «**Enter a number:**» введете число **67**:

```
Let's play!
Enter a number: 67 I have 68
I'm winner!
Press any key to continue...
```

Чому він весь час виграє? Давайте розглянемо рядок

```
cout<<"I have "<<i+1<<"\n";
```

У ній виводиться значення змінної **i**, значення якої ви ввели з клавіатури, збільшене на **1**, тобто комп'ютер завжди виводить число, на **1** більше введеного вами з клавіатури.

Якщо в цій команді замінити вираз **i+1** на вираз **i-1**, то вигравати завжди будете ви, адже число, виведене комп'ютером, завжди буде на одиницю менше за введене вами з клавіатури.

На завершення хочемо звернути вашу увагу на оператори **+** (плюс) та **-** (мінус). Вони використовуються для додавання та віднімання. У мові C++, також є оператор для ділення **/**. Ця інформація допоможе вам при виконанні домашнього завдання, а більш докладно ми поговоримо про оператори в наступних уроках.

10. Літерали

Літерали (literals) — це фіксовані значення, які програма не в змозі змінювати. Для кожного типу в C++ є літерали, включно з символічним і булевським типами, числа цілі та з рухомою крапкою.

Деякі приклади:

5	цілочисельний літерал – int
5l	l або L означає long
true	логічний літерал – bool
5.0	літерал з рухомою точкою, розуміється як double
5.0f	f або F – з рухомою точкою, розуміється як float
0.3e-2	літерал з рухомою точкою double, е або Е відокремлюють експонентну частину
'd'	символьний літерал
«Visual»	рядковий літерал – це набір довільних символів, виділений лапками. Компілятор сприймає його саме як набір символів і ніяк обробляти його не збирається, навіть якщо в лапках виявляться якісь ключові слова й операції.

Приклади коду, що містить літерали.

```
// "abrakadabra" – рядковий літерал
// '\n' – символний літерал
cout<<"abrakadabra"<<'\n';
int a = 2; // 2 – літерал типу int
```

На цьому виклад матеріалу першого уроку можна вважати завершеним. Настійно рекомендуємо вам виконати домашнє завдання, описане в наступному розділі. Бажаємо успіху!

11. Домашнє завдання

- Напишіть програму, яка виводила б на екран текстову таблицю (використовуйте escape-послідовності):

X	Y	X AND Y	X OR Y	NOT X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Малюнок 25

- Дана діагональ телевізора в дюймах. Написати програму, що визначає цю ж діагональ у сантиметрах (1 дюйм = 2.54 сантиметрів).

Нижче представлений приклад для перевірки програми на коректність.

Вхідними даними є:
Діагональ телевізора (inch) – > 42

Вихідні дані:
Діагональ телевізора: 107 см.

- Ємність акумулятора смартфона становить **N** мАг. Написати програму, що визначає мінімальну ємність переносного зарядного пристрою (*powerbank*), якщо необхідні 3 повні заряди смартфона.

Нижче представлений приклад для перевірки програми на коректність.

Вхідними даними є:

Ємність акумулятора смартфона (мАг) -> 3000

Вихідні дані:

Мінімальна ємність: 9000 мАг.

- За один день хом'ячок з'їдає **K** грам корму. Написати програму, що визначає закупівлю корму в кілограмах на 30 днів.

Нижче представлений приклад для перевірки програми на коректність.

Вхідними даними є:

Витрата корму за 1 день (г) -> 20

Вихідні дані:

Обсяг корму на 30 днів: 0.6 кг.

STEP IT Academy, www.itstep.org

Усі права на захищенні авторським правом фото, аудіо та відеоформати, фрагменти яких використані в матеріалі, належать їхнім законним власникам. Фрагменти творів використовуються з ілюстративною метою в обсязі, виправданому поставленим завданням, в межах навчального процесу і в навчальних цілях Відповідно до ст. 21 і 23 Закону України «Про авторське право й суміжні права». Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає шкоди нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора та правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними, що незахищені авторським правом аналогами, і як такі відповідають критеріям сумлінного і чесного використання.

Усі права захищені. Повне або часткове використання матеріалів заборонено. Узгодження використання творів або їхніх фрагментів проводиться з авторами і правовласниками. Узгоджене використання матеріалів можливе лише за умов згадування джерела.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством України.