МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи

**«Створення консольного застосунку мовою C++
у Microsoft Visual Studio 2019»**

з курсу «Алгоритмізація та програмування»
для студентів спеціальності 124 «Системний аналіз»
і курсу «Інформатика і програмування»
для студентів спеціальності 186 «Видавництво і поліграфія»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 26.10.2022 р.

Харків
НТУ «ХПІ»
2023

Методичні вказівки до виконання лабораторної роботи «Створення консольного застосунку мовою C++ у Microsoft Visual Studio 2019» з курсу «Алгоритмізація та програмування» для студентів спеціальності 124 «Системний аналіз» і курсу «Інформатика і програмування» для студентів спеціальності 186 «Видавництво і поліграфія» / Уклад. М. І. Безменов. – Харків : НТУ «ХПІ», 2023. – 36 с.

Укладач М. І. Безменов

Рецензент О. Ю. Чередниченко

Кафедра системного аналізу та інформаційно-аналітичних технологій

ВСТУП

Сучасне програмування орієнтоване не написання програм на папері, а розробку їх у середовищі програмування з налаштунням у діалоговому режимі. У зв’язку із цим знання тільки безпосередньо мови програмування зараз є недостатнім – програміст повинен знати середовище програмування, уміти користуватися тими сервісними засобами, які воно надає.

Метою даної лабораторної роботи є освоєння середовища розробки Microsoft Visual 2019 і отримання початкових навичок у розробці консольного застосунку мовою C++.

1. ТЕОРЕТИЧНІ ОСНОВИ

1.1. Створення проєкту

При запуску Microsoft Visual Studio 2019 відкривається її стартова сторінка, яка, зокрема, може бути використана для створення нового проєкту. Вигляд стартової сторінки наведено на рис. 1.1.

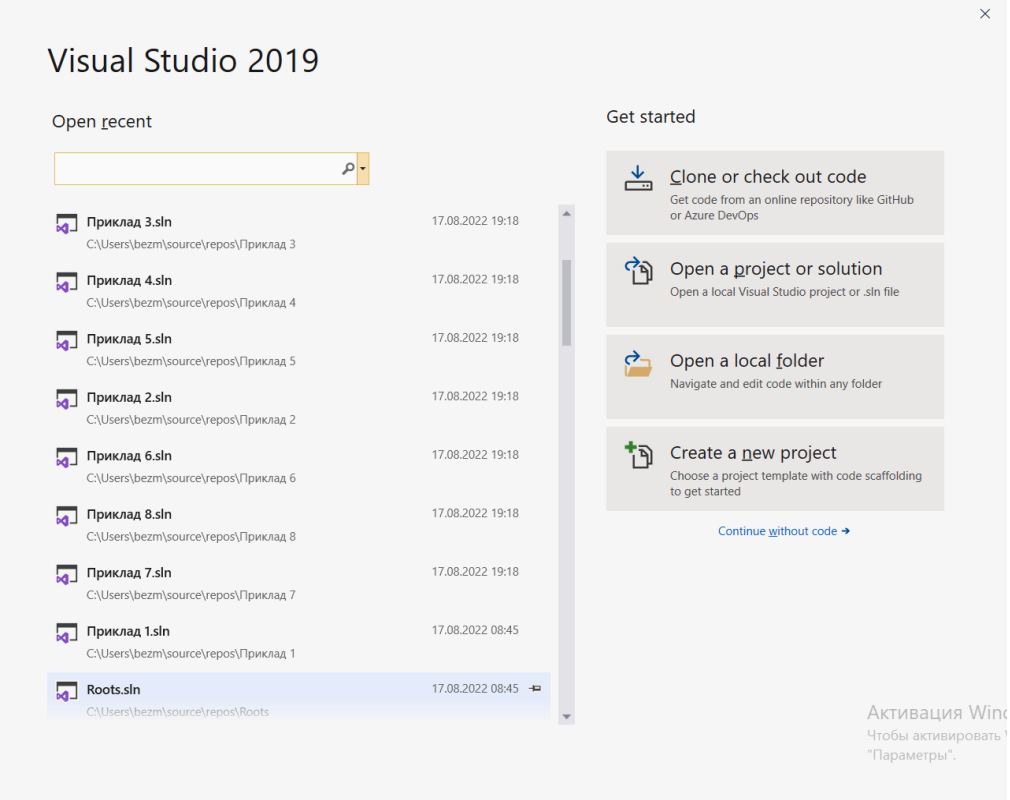


Рис. 1.1. Стартова сторінка

Стартова сторінка при вже працюючій Visual Studio може бути завантажена примусово за командою головного меню File ►Start Window (Файл ►Стартова сторінка). Її вигляд відрізняється від виду, наведеного на рис. 1.1, тільки тим, що назва Visual Studio 2019 замінюється на What would you like to do (Що б ти хотів виконати).

За допомогою стартової сторінки можна виконувати ряд дій, пов'язаних з початком роботи в середовищі. Зокрема, у лівій частині стартової сторінки відбивається вікно з історією відкриття проєктів, рішень тощо. Як результат – можна, наприклад, виконати повторне відкриття раніше розроблюваного проєкту. Оскільки список в історії може бути досить довгим, це вікно постачається смугою вертикального скролінгу, а над ним додатково вбудовано віконце пошуку. У правій частині стартової сторінки наводиться список для вибору початкової дії, зокрема, Open a project or solution (Відкрити проєкт або рішення) та Create a new project (Створити новий проєкт).

При виборі дії Create a new project стартова сторінка отримує вміст, що наведений на рис. 1.2.

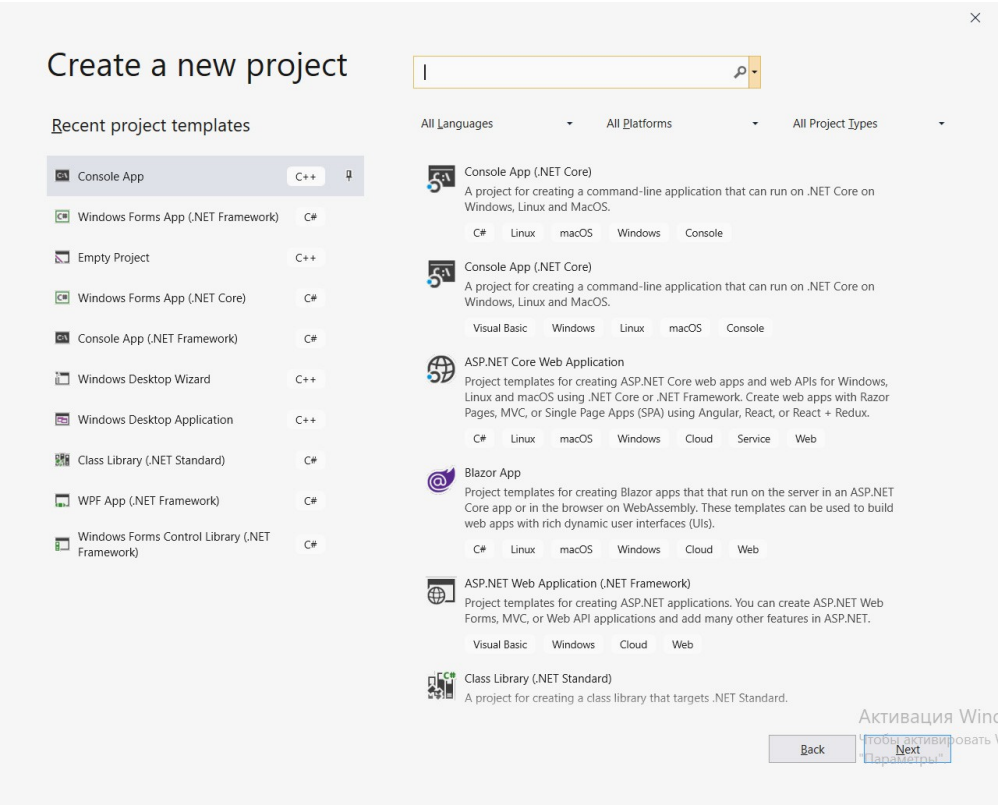


Рис. 1.2. Вигляд стартової сторінки при відкритті нового проєкту

У лівій частині сторінки в такому разі відбивається перелік останніх шаблонів проєктів, які використовувалися в середовищі проєктування, а в правій – повний список можливих шаблонів. При цьому для відбиття повного списку можна встановити фільтри, що дозволяють вибрати мову програмування, використовувану платформу та тип проєкту.

Для створення програми з використанням мови C++ можна вибрати один з шаблонів Console App C++ (Консольний застосунок C++) або Empty Project C++ (Порожній проєкт C++) і натиснути кнопку Next (Далі), у результаті чого відкривається вікно конфігурування нового проєкту (рис 1.3), у якому слід вибрати ім’я проєкту (Project name) і папку для його збереження (Location), після чого слід натиснути кнопку Create (Створити). Якщо у вікні встановлений прапорець Place solution and project in the same directory (Розмістити рішення та проект в одному каталозі), то можна задати ім’я рішення і папки для збереження рішення і папки відповідного проєкту. Інакше ж рішення записується у папку проєкту отримує ім’я, що збігається з його ім’ям.

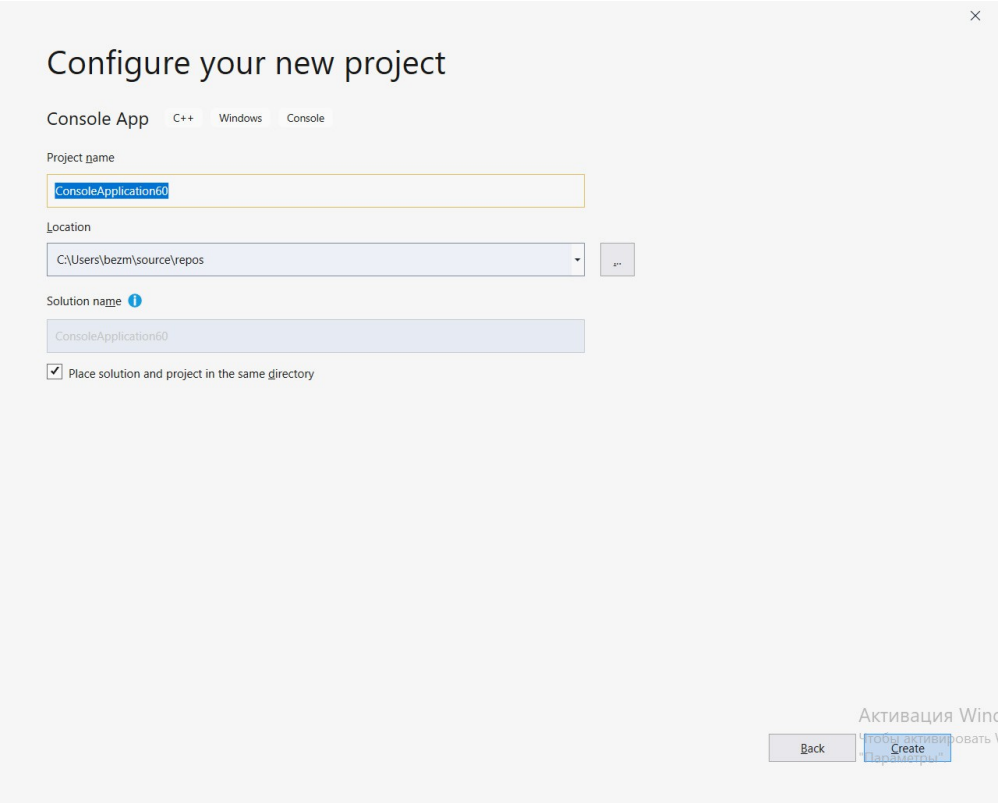


Рис. 1.3. Вікно конфігурування нового проєкту

Після вибору посилання Console App C++ завантажується середовище програмування з головним меню, панелями з інструментальними кнопками та декількома вікнами. Найбільшим з цих вікон є вікно коду, у яке при такому режимі створення проєкту завантажується заготівка уже діючої програми; рис. 1.3. Розробник застосунку коригує цей код відповідно до розв'язуваної задачі.

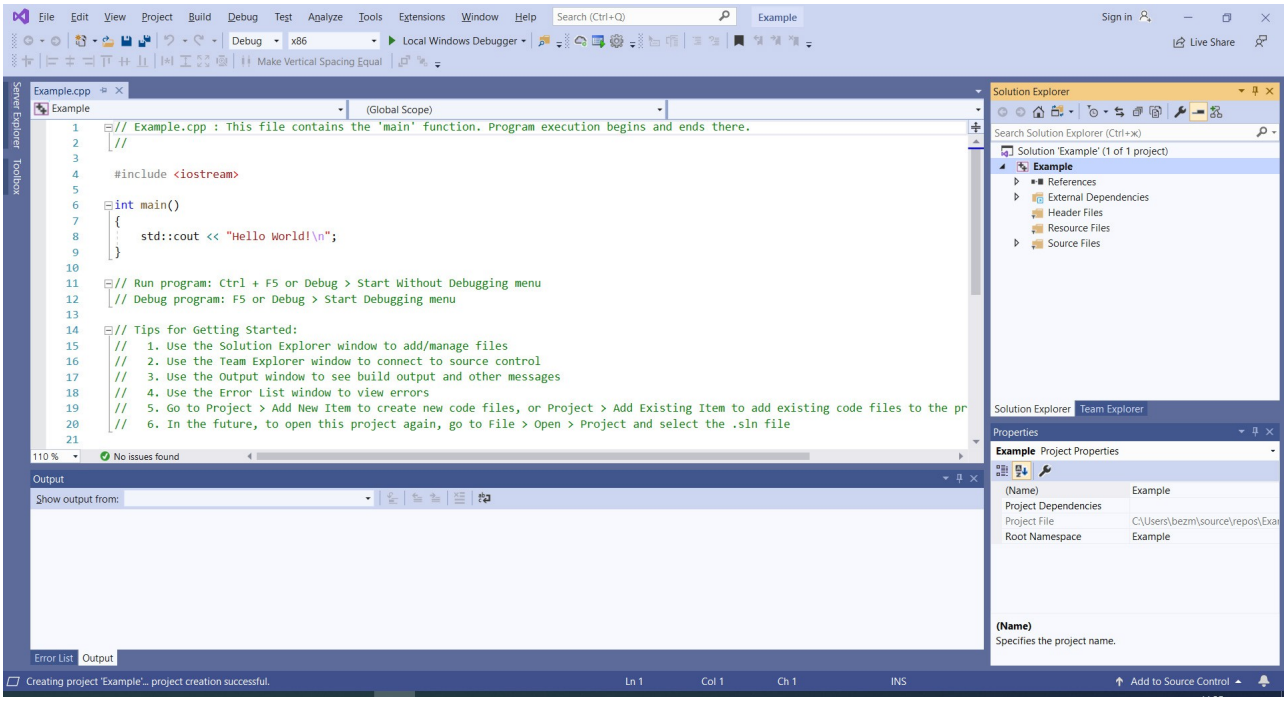


Рис. 1.3. Середовище програмування із завантаженою автоматично створеною заготівкою програми

Особливу роль виконує вікно Solution Explorer (Оглядач рішень), у якому у вигляді дерево відображається структура рішення (папки й файли). Зокрема, у цьому вікні вміст проєкту також відображається у вигляді папки (у даному випадку їй було присвоєно ім'я Example), яка містить вкладені папки. Папка External Dependencies (Зовнішні залежності) відображає файли, які не додані явно до проекту, але використовуються у файлах вихідного коду (наприклад, включені за допомогою директиви #include). Зазвичай у цій папці є заголовні файли стандартної бібліотеки, що використовуються в проекті. Папка Header Files (Заголовні файли) містить так звані заголовні файли, папка Resource Files (Файли ресурсів) містять файли ресурсів (наприклад,

рисунки), папка Source Files (Файли коду) призначена для файлів вихідного коду (з розширенням .cpp).

При додаванні до проєкту (рішення) нових складових вони автоматично відбиваються у вікні оглядача рішень. Додавання нових складових (як і знищення або перейменування існуючих) здійснюється за допомогою контекстного меню, яку відкривається за кліком правою кнопкою миші у вікні оглядача рішень.

Створення нового проєкту можна здійснити й вибравши на стартовій сторінці шаблон Empty Project C++ (Порожній проєкт C++). У такому разі після конфігурування і створення проєкту екран набуває вигляду, наведеного на рис. 1.4.

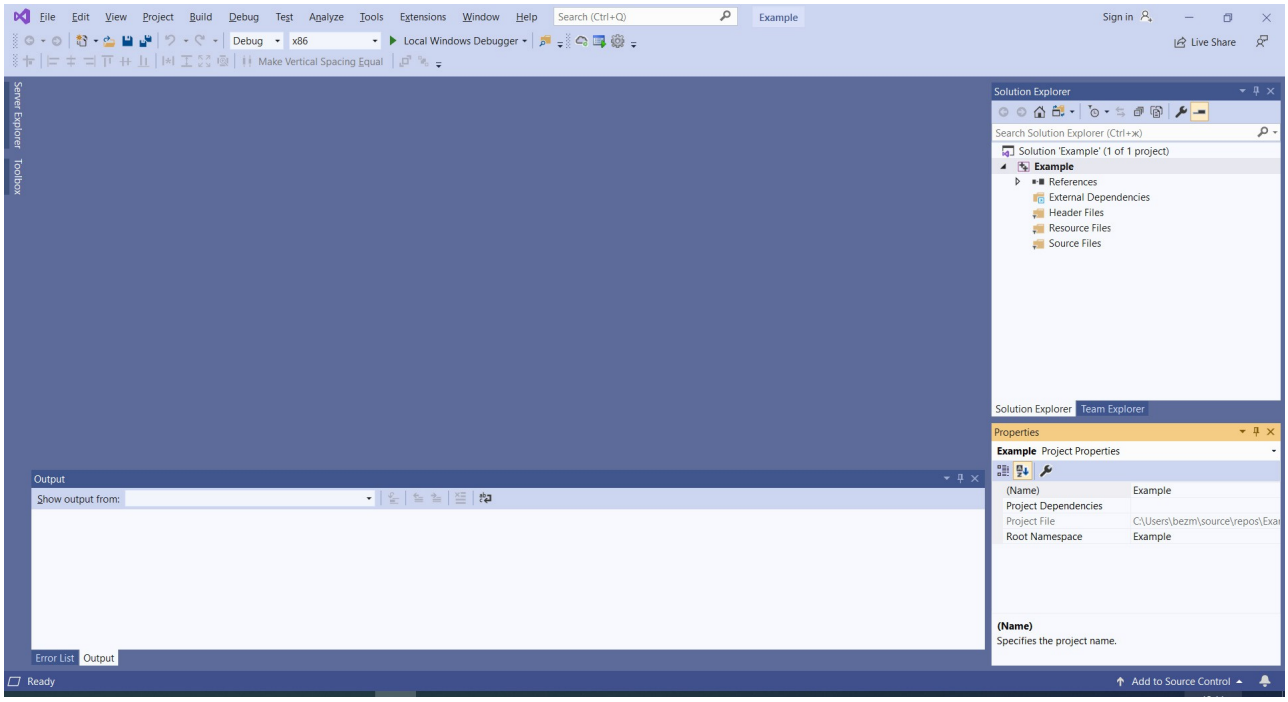


Рис. 1.4. Вигляд середовища програмування у випадку порожнього проєкту

До порожнього проекту необхідно підключити файл з програмним кодом, існуючий або новий. Для створення і підключення нового файлу потрібно у вікні Solution Explorer (Оглядач рішень) клікнути правою кнопкою миші над ім’ям проєкту (у цьому випадку Example) або папкою Source Files (Файли коду) та вибрати в контекстному меню пункт Add –> New Item (Додати ►Новий елемент). У результаті відкривається вікно Add New Item (Додавання нового елемента), наведене на рис. 1.5. Вікно Add New Item можна відкрити також натисканням комбінації

клавіш Ctrl+Shift+A або через головне меню за командою Project ►Add New Item... (Проєкт ►Додати новий елемент...).

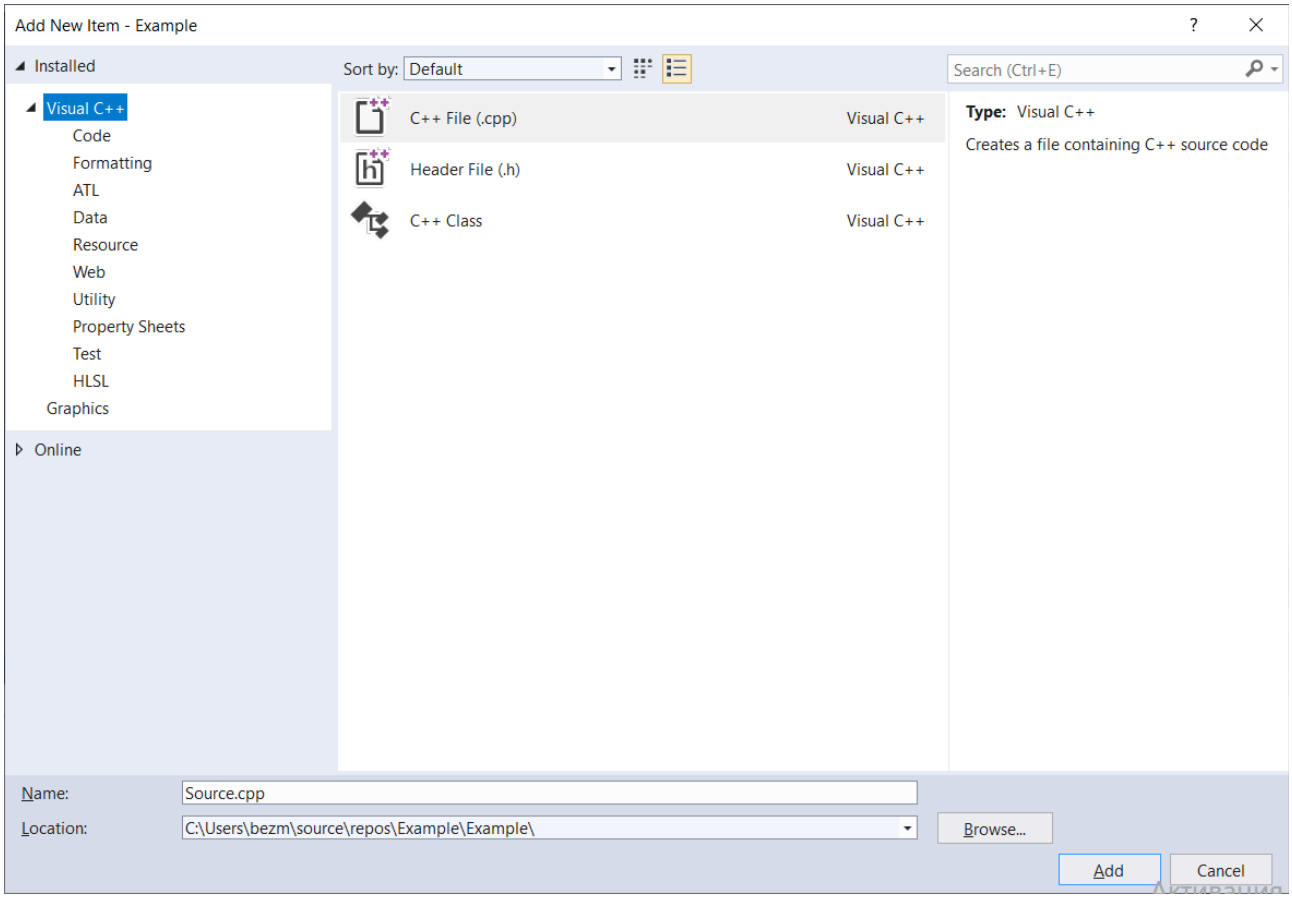


Рис. 1.5. Вікно додавання нового елементу

У цьому вікні потрібно вибрати Visual C++ ►C++ File (.cpp), після чого вказати ім’я cpp-файлу (Name) і його Розташування (Location) і натиснути кнопку Add (Додати). За умовчанням cpp-файлу надається ім’я Source, Source1, Source2 і т. д. залежно від наявності у папці проєкту cpp-файлів із такими іменами (рекомендується не використовувати знеособлені імена, які надаються за умовчанням, а іменувати файли, виходячи із змісту задачі). У результаті автоматично створиться файл із набраним ім’ям (наприклад, Roots) та розширенням .cpp, а також буде відкрито вікно для написання програмного коду (рис. 1.6). Саме в цьому вікні необхідно писати весь код програми (за умови, що програма є однофайловою) і відсутні власні заголовні файли.

Якщо не використовувати стартову сторінку, для створення нового проекту можна скористатися головним меню, виконавши команду File

►New ►Project (Файл ►Створити ►Проєкт). Ця ж команда виконується при натисканні комбінації клавіш Ctrl+Shift+N або за кліком лівою кнопкою миші над інструментальною кнопкою  (New Project).

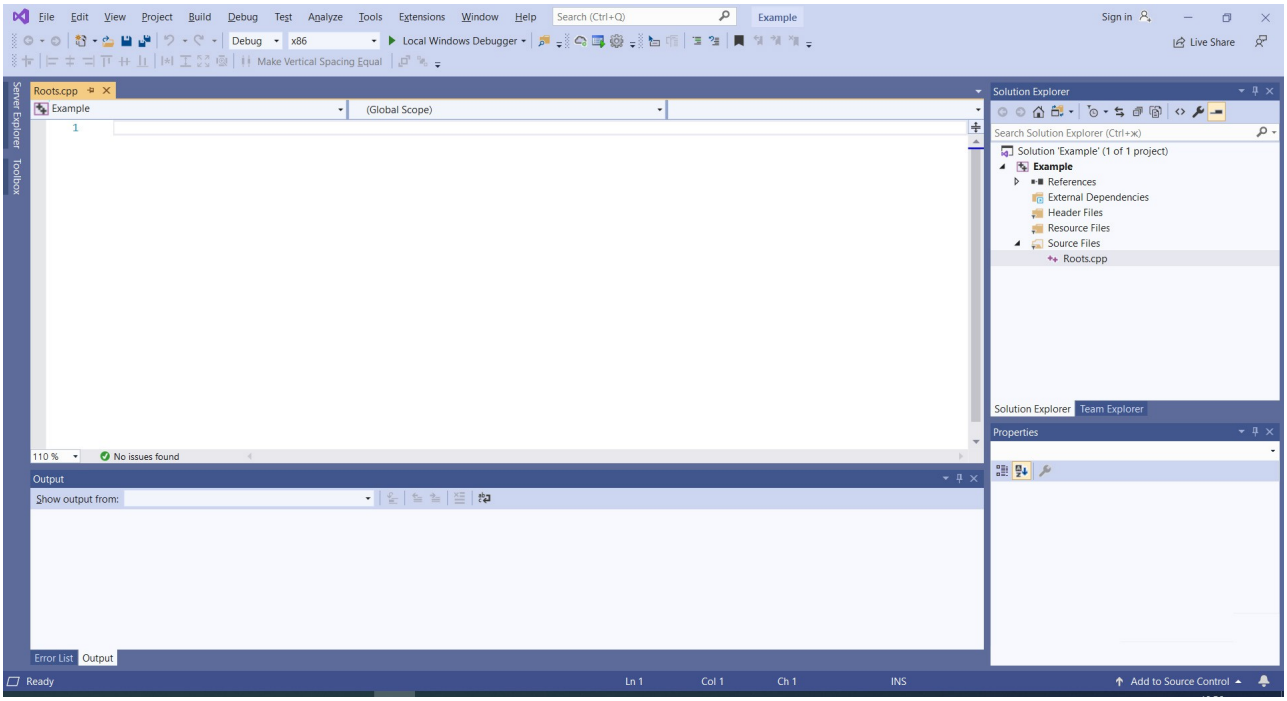


Рис. 1.6. Вигляд інтегрованого середовища із завантаженим проєктом, до якого добавлений порожній файл Roots.cpp

Файл з кодом програми можна видалити з проєкту. Для цього необхідно в оглядачі рішень відкрити папку Source Files і клікнути правою кнопкою миші над ім’ям cpp-файлу, у результаті чого відкриється контекстне меню, в якому слід виконати команду Remove (Видалити). Аналогічно можна здійснити видалення інших складових рішення або проєкту.

1.2. Робота з вікном коду

Вікно коду є звичайним вікном багаторядкового текстового редактора, і саме в ньому здійснюється набирання тексту програми.

Якщо програма є багатомодульною, кожен модуль може бути завантажений на окремій сторінці вікна коду, причому кожна сторінка має закладку з іменем завантаженого на неї модуля (на рис. 1.6 завантажений тільки один, поки що порожній файл, а саме файл Roots.cpp).

Робота з вікном коду не має ніяких особливостей відносно роботи в будь-якому іншому редакторі:

- набирання тексту здійснюється в позиції, де знаходиться курсор;
- переміщення курсору здійснюється кліком мишкою над потрібним місцем вікна або за допомогою клавіш керування курсором, а саме:
  - стрілка ліворуч, стрілка праворуч, стрілка вгору, стрілка вниз – переміщення курсору на одну позицію у відповідному напрямі;
  - Home – переміщує курсор на початок поточного рядка;
  - End – переміщує курсор на кінець поточного рядка;
  - Page Up – переміщує курсор на одну сторінку екрану вгору;
  - Page Down – переміщує курсор на одну сторінку екрану вниз;
  - Ctrl+стрілка ліворуч – переміщує курсор на одне слово ліворуч (до найближчого символу-роздільника);
  - Ctrl+стрілка праворуч – переміщує курсор на початок наступного слова (усі символи-роздільники пропускаються);
  - Ctrl+стрілка вгору – переміщує редакторське вікно відносно тексту на один рядок угору (візуально текст у редакторському вікні зсувається на один рядок вниз);
  - Ctrl+стрілка вниз – переміщує редакторське вікно відносно тексту на один рядок вниз (візуально текст у редакторському вікні зсувається на один рядок угору);
  - Ctrl+Home – переміщує курсор на початок тексту;
  - Ctrl+End – переміщує курсор на кінець тексту;
  - Ctrl+Page Up – переміщує курсор на перший рядок тексту у вікні редактора, не змінюючи його положення по горизонталі;
  - Ctrl+Page Down – переміщує курсор на останній рядок тексту у вікні, не змінюючи його положення по горизонталі;
- натискання клавіші Backspace приводить до знищення одного символу безпосередньо перед курсором;
- натискання клавіші Delete приводить до знищення одного символу безпосередньо за курсором;

- клавіша Ctrl+Backspace знищує всі символи поточного слова ліворуч від курсору до найближчого символу-роздільника;
- клавіша Ctrl+Delete знищує всі символи праворуч від курсору до першого символу наступного слова;
- відновити текст до останнього редагування можна за допомогою команди меню Edit ►Undo (Правлення ►Скасувати) або за натисканням клавіші Ctrl+Z (цю операцію можна робити декілька разів поспіль);
- повернути текст після відновлення результатів редагування (у тому числі декілька разів поспіль) можна за допомогою пункту меню Edit ►Redo (Правлення ►Повернути) або натисканням комбінації клавіш Shift+Ctrl+Z;
- виділення тексту здійснюється зміщенням курсору при натиснутій лівій кнопці мишки або зміщенням курсору за допомогою клавіш керування ним при натиснутій клавіші Shift (або кліком мишкою над позицією, де закінчується виділення, при натиснутій клавіші Shift);
- виділення всього вмісту вікна коду виконується натисканням комбінації клавіш Ctrl+A;
- натискання будь-якої символьної клавіші в разі наявності виділення приводить до заміни виділеного тексту уведеним символом;
- натискання клавіш Backspace та Delete приводить до знищення всього виділеного тексту, а не одного символу, як це буває в разі відсутності виділення;
- виділений текст може бути скопійований у буфер обміну, для чого треба вибрати пункт меню Edit ►Copy (Правлення ►Копіювати) або натиснути клавішу Ctrl+C (або Ctrl+Insert);
- для копіювання виділеного тексту з його знищенням (вирізання) служить пункт меню Edit ►Cut (Правлення ►Вирізати) або клавіша Ctrl+X (або Ctrl+Delete);
- вставка тексту, що знаходиться в буфері обміну, здійснюється вибором пункту меню Edit ►Paste (Правлення ►Вставити) або натисканням клавіші Ctrl+V (або Shift+Insert);
- пошук або пошук із заміною забезпечується командами, доступними у опції Edit ►Find and Replace (Правлення ►Пошук і заміна).

1.3. Компіляція та компонування

Компілятор (Compiler) – це особлива програма, для якої як вхідні дані виступають програми, написані мовою високого рівня, і на виході якої також утворюються програми, але вже написані машинною мовою.

Програми, отримані на виході компілятора, називаються ***об'єктними програмами***, або ***об'єктними кодами***. Особливістю таких програм є те, що вони не можуть бути відразу виконані, оскільки до того необхідно здійснити підключення до них деяких службових підпрограм.

Остаточне опрацювання програми, точніше об'єктного коду, здійснює так звана ***програма-компонувальник*** (інакше ***редактор зв'язків***, Linker). Саме компонувальник створює ***машинний код***, який і виконується комп'ютером при розв'язанні задачі. Компонувальник поєднує об'єктний код програми з об'єктними кодами службових підпрограм, які використовуються даною програмою.

1.4. Структура однофайлової програми

Однофайлова програма мовою C++ являє собою послідовність препроцесорних директив, описів і визначень глобальних об'єктів і функцій (див. приклад на рис. 1.7).

Препроцесорні директиви управляють перетворенням тексту програми до її компіляції. Це рядки, що починаються із символу #, слідом за яким йде ім'я директиви (наприклад, #define або #include) і додаткова інформація. Препроцесор виконує пошук у тексті програми рядків, що починаються із символу #, і перетворення тексту програми відповідно до вмісту цього рядка (наприклад, визначаються деякі імена).

Найчастіше за допомогою препоцесорних директив здійснюється підключення бібліотек для розширення функціональних можливостей коду. ***Бібліотека*** – це набір функцій (у тому числі зі стандартних бібліотек), а також визначених поза програмою змінних та констант, які можуть бути використані в програмі й зберігаються у відкомпільованому вигляді.

Отриманий у результаті роботи препроцесора повний текст програми оброблюється компілятором, який створює ***об'єктний файл*** – прог-

раму машинною мовою, яка не може бути виконана через відсутність зв'язків з компонентами, що перебувають в інших файлах. Виконуваний модуль (exe-файл) будується з об'єктних файлів (obj-файлів) компонувальником, що збирає відкомпільований текст програми і функції з бібліотек в одну виконувану програму.

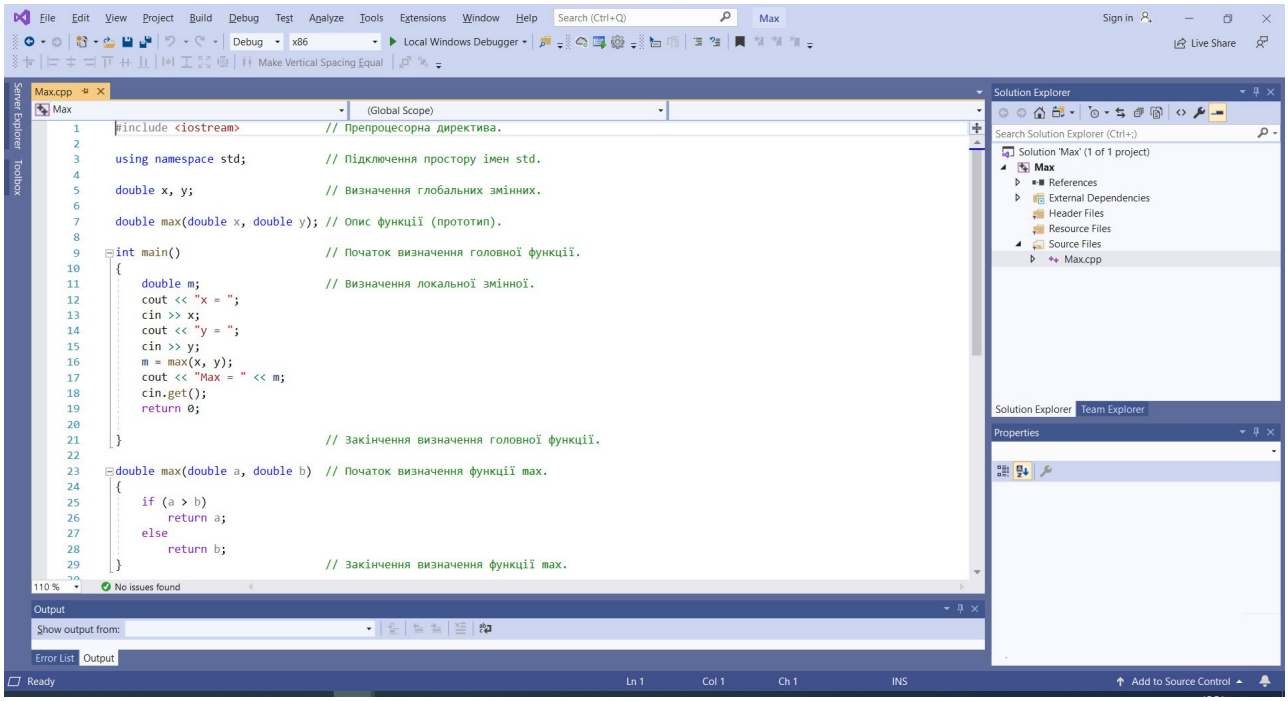


Рис. 1.7. Структура однофайлової програми

Визначення вводять функції та об'єкти, а описи повідомляють компілятор про властивості й імена тих об'єктів і функцій, які визначені в інших частинах програми (нижче по тексту або в іншому файлі). ***Функції*** визначають потенційно можливі дії програми і є самостійними одиницями. Кожна функція має ім'я та список аргументів, забраний у круглі дужки. Дужки вказуються навіть при відсутності аргументів, що дозволяє відрізнити ім'я функції від імен змінних. Далі у фігурних дужках записується ***тіло функції***, що являє собою послідовність ***інструкцій***, кожна з яких у мові C++ завершуються символом «крапка з комою».

1.5. Налаштування програми

Процес усунення помилок у програмі називається ***налаштуванням*** (debugging), а самі помилки в програмі досить часто називають ***жучками***

(bugs). Серед помилок, які допускаються в програмі, виділяють синтаксичні помилки, помилки часу виконання і логічні помилки.

Синтаксичні помилки – це помилки, пов'язані з порушенням ***синтаксису*** (тобто правил граматики) мови програмування. Прикладом такої помилки, що часто зустрічається в програмах, є пропуск символу «крапка з комою», яким завершується кожна інструкція в програмах, написаних мовою C++. При виявленні синтаксичної помилки компілятор видає ***повідомлення про помилку***, вказуючи її передбачуване місце розташування і пояснюючи можливий її зміст. Звісно, природа помилки може відрізнятися від тієї її інтерпретації, яку робить компілятор, тим більше що одна синтаксична помилка може призвести до того, що компілятор видасть декілька повідомлень про помилки, зумовлені, проте, наявністю лише першої з них. Може бути і протилежний випадок, коли одна синтаксична помилка приховує від компілятора інші помилки, що наявні далі в тексті програми.

Досить часто компілятор виводить ***попереджувальні повідомлення*** про деякі дещо незвичні конструкції у програмі. У багатьох випадках програмісти звертають увагу тільки на повідомлення компілятора про наявність помилок (Errors), не реагуючи на попередження (Warnings). У той же час попереджувальні повідомлення можуть свідчити про помилки, у зв'язку з чим на них варто обов'язково звертати увагу і вносити відповідні виправлення. Наприклад, такою помилкою є використання в обчислюваних виразах локальних змінних, які раніше не отримали значення. За умовчанням у такому разі діагностується помилка, але можливий режим компіляції, коли в такому випадку виводиться тільки попереджувальне повідомлення, на яке безумовно необхідно реагувати виправлення тексту програми. В ідеальному випадку попереджувальні повідомлення відсутні.

Для запуску процесу компіляції cpp-файлу, ім'я якого виділене в Solution Explorer або завантажене на активну сторінку вікна коду, слід виконати команду Build ►Compile (Побудування ►Компіляція), якій відповідає гаряча клавіша Ctrl+F7, або команду Compile (Компіляція) у контекстному меню, що відкривається при кліку правою клавішею мишки над іменем cpp-файлу в Solution Explorer. Формально файл, що містить

програмний код, може бути направлений і на обробку компонувальником, для чого слід виконати команду Build ►Build ім'я файлу (Побудування ►Побудування ім'я файлу). При цьому перед компонуванням виконується попередня компіляція файлу. Компонування виконуватиметься тільки в разі успішної компіляції. Для компіляції програми разом з компонуванням слід обрати пункт меню Build ►Build Solution (Побудування ►Побудування розв'язку) – гаряча клавіша Ctrl+Shift+B (залежно від варіанту інсталяції за умовчанням може бути встановлена клавіша F6).

Якщо створений програмний код є некоректним, то по закінченню процесу компіляції і компонування у вікні Error List (Список помилок) буде виведено інформацію про виявлені Помилки (Errors), Попередження (Warnings) та Повідомлення (Messages); рис. 1.8.

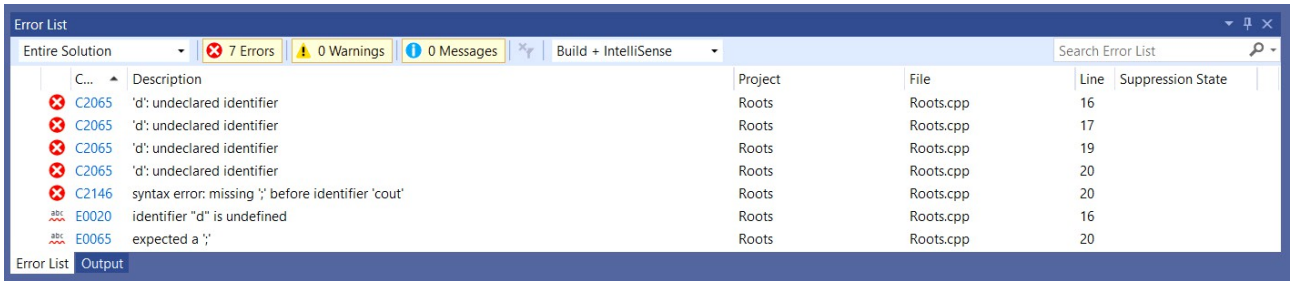


Рис. 1.8. Вікно Error List з інформацією про помилки

Функціональні можливості технології IntelliSense дозволяють забезпечити виведення у вікні Error List рекомендацій по виправленню помилок. Дозволити (заборонити) виведення цих рекомендацій можна за кліком правою клавішею мишки у вікні Error List, встановивши (скинувши) прапорець Show IntelliSense Errors у віконці, яке відкривається при цьому.

Процес компіляції відображується і у вікні Output (Виведення), де також виводиться інформація про помилки і попередження в разі їх наявності (рис. 1.9).

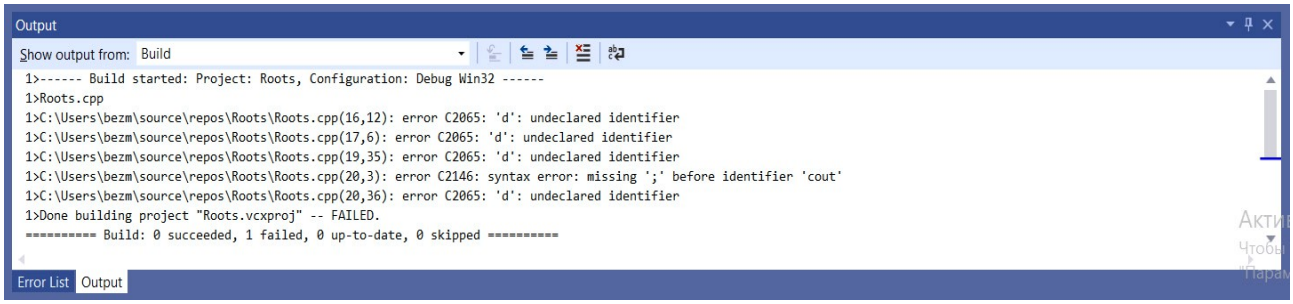


Рис. 1.9. Вікно Output з інформацією про помилки

Можливим є також вибір пункту меню Debug ►Start Debugging (Налаштування ►Почати налаштування). Ця ж команда виконується за натисканням гарячої клавіші F5 або за кліком мишкою над інструментальною кнопкою  на панелі Standard (Стандартна). Можна також натиснути гарячу клавішу Ctrl+F5, що відповідає виконанню команди меню Debug ►Start Without Debugging (Налаштування ►Почати без налаштування). У цих випадках, насправді, мова йде про запуск програми на виконання. Але перед виконанням програми, якщо в її тексті здійснювалася корекція, будуть автоматично виконані компіляція програми та компонування. У разі відсутності синтаксичних помилок і помилок компонування програма буде автоматично запущена. Якщо ж помилки будуть виявлені, на екран виводиться зображене на рис. 1.10 діалогове віконце з попередженням про появу помилки компонування і запитанням про згоду на виконання останнього успішного варіанту компонування.

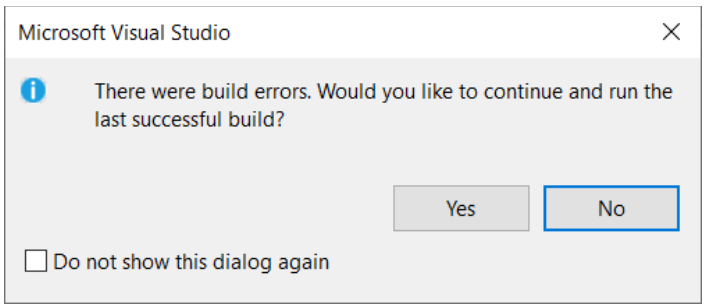


Рис. 1.10. Діалогове віконце помилки компонування

Якщо в цьому віконці встановити прапорець Do not show this dialog again (Не показувати цей діалог), то в подальшому воно не буде виводитися, а програма виконуватиметься навіть при наявності

помилок (точніше виконуватиметься останній успішний варіант компонування). Тому цей прапорець краще на встановлювати. Відновлення виведення вказаного діалогового віконця в разі його відміни здійснюється виконанням команди меню Debug ►Options... (Налаштування ►Параметри...). Далі потрібно розвернути список Projects and Solutions (Проєкти і розв'язки), в якому вибрати пункт Build and Run (Побудування та запуск). Після вибору цього пункту в правій частині вікна Options виведеться декілька списків, що розкриваються. Одним з них є список On Run, when projects are out of date: (Запуск застарілого проєкту:). Він має три пункти:

Prompt to launch – Підказувати перед запуском

Do not launch – Не запускати

Launch old version – Запуск старої версії

Далі потрібно вибрати в цьому списку пункт Prompt to launch і підтвердити вибір кліком мишкою над кнопкою OK (можна також рекомендувати вибір пункту Do not launch). Перед натисканням кнопки OK можна встановити прапорець For new solutions use the currently selected project as the startup project (Для нових розв'язків використовувати вибраний варіант запуску проєкту).

Для виведення вікна Options замість команди Debug ►Options можливе також використання команди Tools ►Options... (Сервіс ►Параметри...).

Відзначимо, що вікна Error List і Output можуть бути закриті. Для відкриття цих вікон потрібно виконати відповідно команди View ►Error List (Перегляд ►Список помилок) і View ►Output (Перегляд ►Виведення). Їм відповідають послідовне натискання клавіш Ctrl+\, E, а також комбінація клавіш Ctrl+Alt+O.

Деякі помилки проявляються тільки під час виконання програми, у зв'язку з чим вони носять відповідну назву – ***помилки часу виконання*** (run-time errors). У термінології C++ такі помилки називаються ***винятками*** (Exceptions). З появою таких помилок програма завершується аварійно з виведенням пояснювального повідомлення. У середовищі поява таких помилок призводить до переривання виконання програми і виведення вікна з інформацією про вид винятку. Прикладами таких

помилок часу виконання є ділення на нуль, одержання великих числових значень, які не можуть бути записані в комірку пам'яті (переповнення). Наприклад, на рис. 1.11 зображене вікно, яке висвітлюється у випадку, коли виконується ділення на нуль при обробці двох цілих значень.

Найбільш неприємними є ***логічні помилки*** – помилки в самому алгоритмі і помилки, спричинені елементарною неуважністю (наприклад, використання в програмі оператора * замість оператора +, задавання неправильного числового значення, використання одного імені змінної замість іншого). Такі помилки компілятор найчастіше за все виявити не може (за винятком випадків, коли вони призводять до порушення синтаксису). Більш того, логічні помилки можуть не проявити себе і при виконання програми.

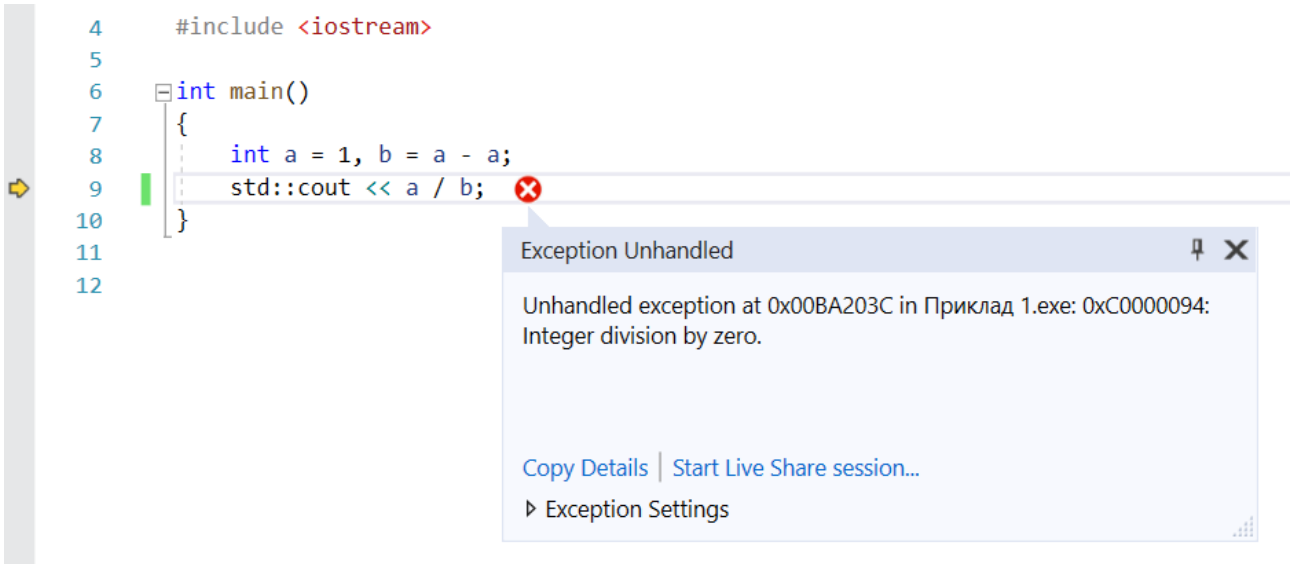


Рис. 1.11. Вікно виникнення виключення

Для виявлення логічних помилок здійснюється ***тестування*** програми, яке виражається в її запуску з кількома характерними наборами вхідних даних і перевірці отриманих результатів. При цьому в жодному разі не можна обмежуватися одноразовою перевіркою програми – повинні бути відстежені всі окремі випадки вхідних даних, з якими програма може зіткнутися. Тільки після перевірки правильності функціонування програми на багатьох наборах даних, як типових, так і тих, що характеризують особливі випадки, можна отримати високу (але не абсолютну) гарантію її правильності.

Найчастіше за все, щоб знайти причину логічної помилки, треба виконати якийсь фрагмент програми, спостерігаючи значення змінних при виконанні кожної команди. Для виявлення більшості логічних помилок використовують так званий налаштувальник (Debugger).

Робота в режимі налаштування – це, насамперед, виконання програми по кроках з відстеженням послідовності виконання інструкцій програми та значень, які набувають змінні після виконання того або іншого кроку.

Для виконання фрагменту програми по кроках можна використовувати такі команди меню:

- Debug ►Step Into (Налаштування ►Крок із заходом у ...) – покрокове виконання програми із заходом у функції з наступним покроковим виконанням їх рядків; аналогом є натискання клавіші F11 або клік мишкою над інструментальною кнопкою  на панелі Debug (Налаштування), яка автоматично висвітлюється в інтегрованому середовищі в разі зупинки виконання програми під час налаштування;
- Debug ►Step Over (Налаштування ►Крок з обходом) – покрокове виконання рядків програми, при якому виклик функції вважається за одну команду, тобто вхід у функції не проводиться (аналогом є натискання клавіші F10 або клік мишкою над інструментальною кнопкою  на панелі Debug);
- Debug ► Step Out (Налаштування ►Вийти) – виконання програми до виходу з поточної функції і зупинка на інструкції, наступній за викликом цієї функції, з залишенням курсору в рядку, в якому було здійснене звертання до функції (аналогом є натискання клавіші Shift+F8 або клік мишкою над інструментальною кнопкою  на панелі Debug).

Можливе також виконання команди Run to Cursor (Виконати до курсору), яка викликається при натисканні комбінації клавіш Ctrl+F10 або за звертанням до контекстного меню кліком правою кнопкою мишки. У цьому разі програма виконується до рядка, в якому знаходиться курсор.

Для налаштування програми використовуються так звані ***точки переривання***. Щоб увести просту (безумовну) точку переривання, достатньо у вікні редактора коду клікнути мишкою на лівій межі вікна навпроти рядка коду, перед виконанням якого потрібно призупинити обчислення

для аналізу проміжних результатів. При цьому навпроти рядка на лівій межі вікна коду з'явиться червона кулька. Якщо тепер запустити програму на виконання, то кожен раз коли керування перейде до рядка, в якому вказана точка переривання, відбуватиметься переривання виконання. Зняти точку переривання можна також кліком мишкою над лівою межею вікна коду. Для встановлення (зняття) точки переривання можна також використовувати гарячу клавішу F9 або команду меню Debug ► Toggle Breakpoint (Налаштування ►Перемикач контрольної точки). Виконання цієї команди встановлює (знищує) точку переривання на рядку, де в момент її виконання знаходиться курсор.

Дія точки переривання дещо аналогічна виконанню до курсору (Ctrl+F10), але перевага точок переривання полягає в тому, що можна одночасно вказати декілька таких точок у різних місцях коду і в різних модулях. Програма в цьому разі виконується до першої точки переривання, яка зустрінеться під час виконання. Після аналізу проміжних результатів можна продовжити виконання програми, натиснувши інструментальну кнопку 

на одній з панелей Standard (Стандартна) або Debug (Налаштування). Можна також натиснути клавіша F5 або виконати команду меню Debug ►Continue (Налаштування ►Продовжити). Слід відзначити, що за умовчанням кнопка 

на панелі Debug (Налаштування) прихована.

Виконання програми може бути припинено примусово. Для цього необхідно натиснути інструментальну кнопку  на панелі Debug (Налаштування). Для переривання виконання програми і її повторного запуску може бути використана інструментальна кнопка  на тій же панелі.

Для перегляду значення змінних у момент зупинки програми під час налаштування достатньо навести курсор мишки на ім'я необхідної змінної в коді програми. Якщо ж є необхідність відслідковувати стан одразу декількох змінних, це можна зробити у вікнах Autos (Автоматичні, краще Видимі), Locals (Локальні), Watch 1 (Контрольне значення 1), приклади наповнення яких зображені на рис. 1.12–1.14. Можна також відкрити вікна Watch 2, Watch 2 і Watch 3.

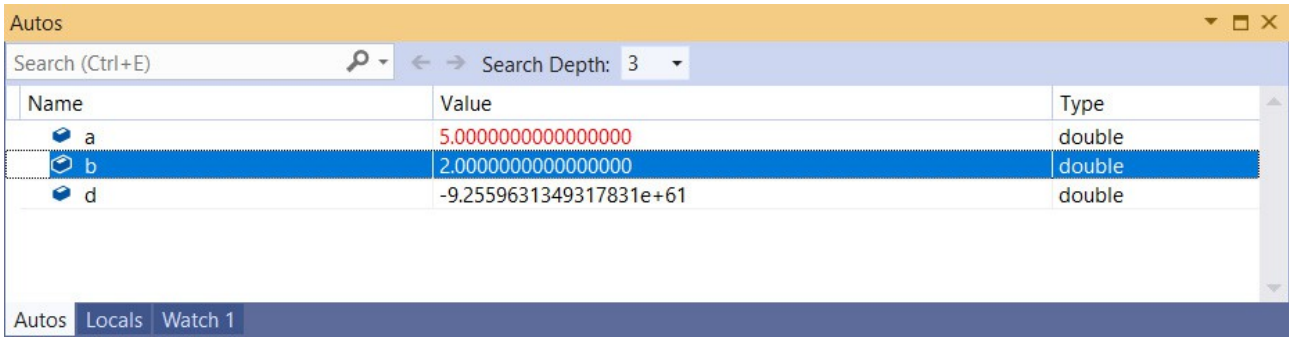


Рис. 1.12. Вікно Autos

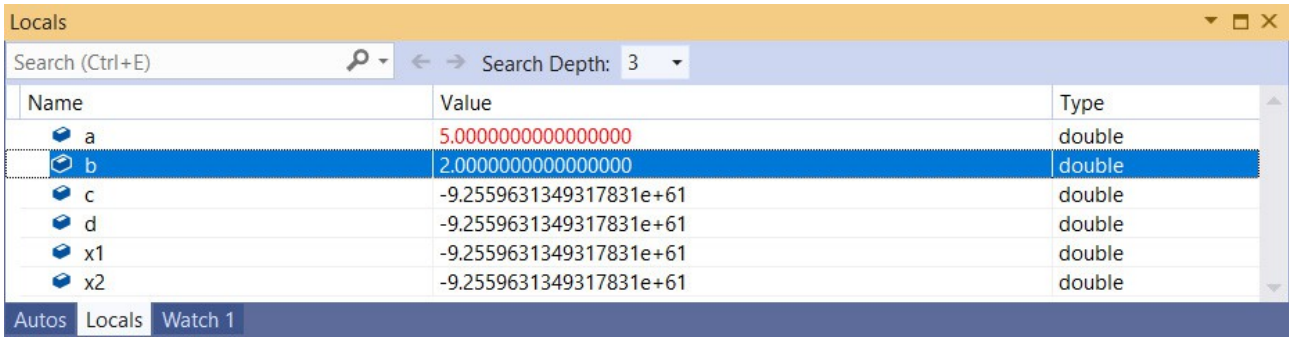


Рис. 1.13. Вікно Locals

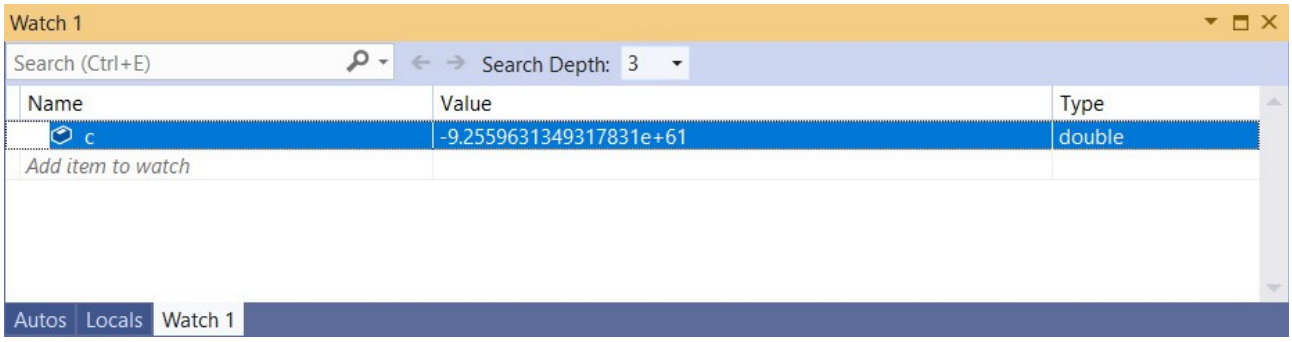


Рис. 1.14. Вікно перегляду стану змінних Watch1

За умовчанням три перші з шести перелічених вище вікон перегляду стану змінних автоматично виводяться в режимі налаштування під вікном коду в лівій частині екрану. У разі відсутності потреби в будь-якому з цих вікон, його можна закрити. Для відкриття такого вікна, необхідно в режимі налаштування вибрати в головному меню опцію Debug ► Windows ►Autos (Налаштування ► Вікна ►Видимі), Debug ►Windows ►Locals (Налаштування ►Вікна ►Локальні) або Debug ► Windows ► Watch ►Watch 1 (Налаштування ► Вікна ►

Контрольне значення ►Контрольне значення 1). Для вікон Watch 2, Watch 3, Watch 4 потрібно вибрати команду, що відповідає номеру вікна.

Вікно Autos – це автоматичне вікно, що показує стан локальних змінних які використовуються в поточний момент. Елементами цього вікна є змінні поточної інструкції (тієї, що виділена жовтою стрілкою на лівій межі вікна коду) і попередньої інструкції (тієї, яка тільки що була виконана). Як тільки програма доходить до виконання коду, що не використовує ці змінні, вони зникають з вікна Autos.

Вікно Locals – це автоматичне вікно, що показує значення усіх локальних змінних (у тому числі формальних параметрів функції). Елементами цього вікна є всі змінні (навіть однойменні), які існують на даному рівні вкладеності блоків.

Контрольовані змінні у вікнах Autos і Locals обирає сам комп'ютер, а не користувач, причому значення виразів у цих вікнах не відображуються Для самостійного вибору змінних і виразів використовуються вікна Watch, яких може бути до чотирьох (Watch1, Watch2, Watch3, Watch 4). Для додавання змінних і виразів у вікна Watch достатньо здійснити клік мишкою в лівій частині порожнього рядка такого вікна і увести ім'я змінної або вираз. Якщо виділити рядок вікна Watch і набрати будь-який текст, то цей текст сприймається як вираз, який заміщує текст, що містився раніше в цьому рядку (для змінних ми маємо перейменування). За кліком правою кнопкою мишки у вікні Watch відкривається контекстне меню, в якому можна вибрати опцію Add Watch (Добавити контрольне значення), у результаті чого у вікні з'являється рядок – копія рядка, над яким здійснювався клік. Далі можна виконати перейменування, додавши тим самим нове контрольне значення. Контрольне значення можна додати у вікно Watch «перетаскуванням» з вікна коду або виділенням рядка в одному з вікон Autos і Locals, відкриттям контекстного меню і виконанням команди Add Watch. Видалення контрольного значення здійснюється натисканням клавіші Delete або за допомогою контекстного меню (опція Delete Watch).

Значення змінних, що відображені в будь-якому з вікон Watch, можна примусово змінювати. Для цього достатньо виділити це значення у вікні і набрати нове. При подальшому виконанні програми в змінній буде

міститися нове значення. Це в складних випадках дозволяє продовжити налаштування програми, не виправляючи помилок вище за її текстом (ніби має місце така ситуація: «Будемо вважати, що змінна *x* має правильне значення. Як виконуватиметься програма в цьому випадку?»).

1.6. Завершення виконання консольного застосунку

Якщо не передбачити ніяких додаткових дій у програмному коді, то виконання застосунку завершується автоматичним закриттям консольного вікна, у яке здійснюється виведення результату. У даному випадку мова йде про виконання exe-файлу, який є результатом роботи компонувальника (він записується у підпапку Debug або Release папки, де міститься проєкт). Таким чином, якщо виведення кінцевого результату, здійснюється не у файл, а на екран, побачити його стає неможливим. У такому разі необхідно передбачити затримку закриття консольного вікна до виконання деякої дії, якою звичайно є натискання будь-якої клавіші.

Звичайно для затримки закриття консолі користуються одним з наступних трьох методів:

1. Виклик функції `system` з параметром `"pause"`:

```
system("pause");
```

Ця функція виводить повідомлення `Для продовження натисніть будь-яку клавішу` і затримує виконання програми. Щоб скористатися цією функцією треба підключити один із заголовних файлів `process.h`, `stdlib.h`, або `cstdlib` (можна скористатися і файлом `iostream`). Це робиться за допомогою рядка

```
#include <cstdlib>
```

записуваного зазвичай на початку файлу з програмним кодом (у кутових дужках зазначається ім’я заголовного файлу). Слід зазначити, що варіант з функцією `system` працює не в усіх операційних системах.

2. Виклик функції `_getch`:

```
_getch();
```

що зчитує один символ, переводячи програму в режим очікування введення символу.

Функція `_getch` оголошена в заголовному файлі `conio.h`, повинен бути підключеним:

```
#include <conio.h>
```

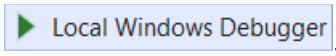
Оскільки при цьому не виводиться ніяке попереджувальне повідомлення, слід про це подбати окремо, наприклад, так:

```
cout << "Press any key to close the application.";
_getch();
```

Варіант затримки закриття консольного вікна за допомогою `_getch` теж працює не в усіх операційних системах.

3. Виклик компонентної функції `get` без параметрів об'єкта `std::cin` для введення одного символу. При цьому перед викликом цієї функції для затримки виконання програми слід звільнити буфер потоку:

```
std::out << "Press any key to close the application.";
int ch;
while (!((ch = std::cin.get()) == '\n' || ch == EOF))
    ;       // Очищення буферу потоку введення.
cin.get();  // Уведення одного символу (затримка).
```

Якщо виконання застосунку запущено без налаштування за командою Debug ► Start Without Debugging (Налаштування ►Почати без налаштування), якій відповідає клавіша Ctrl+F5, то середовище перед завершенням роботи програми автоматично затримує консольний екран. Запуск же програми за командою Debug ►Start Debugging (Налаштування ►Почати налаштування), якій відповідає натискання гарячої клавіші F5 або клік мишкою над інструментальною кнопкою 

, то Visual Studio не затримує консольне вікно.

У Visual Studio 2019 незалежно від методу запуску програми з середовища розробника за умовчанням перед завершенням виконання здійснюється затримка консольного вікна з висвітлення повідомлення про завершення роботи і необхідність натискання будь-якої клавіші. Якщо ж при запуску за натисканням клавіші F5 затримка по завершенні програми не потрібна, треба виконати команду меню Debug ►Options (Налаштування ►Параметри), у вікні Options (Параметри) вибрати

Debugging ►General (Налагодження ►Загальні), після чого встановити прапорець Automatically close the console when debugging stop (Автоматично закривати консоль після зупинки налагодження).

2. ПРИКЛАД СТВОРЕННЯ НОВОГО КОНСОЛЬНОГО ЗАСТОСУНКУ ТА ЙОГО НАЛАШТУВАННЯ

Розглянемо можливу послідовність дій по налаштуванню програми на прикладі задачі розробки застосунка для відшукання коренів квадратного рівняння

$$ax^2 + bx + c = 0,\ a \neq 0.$$

Умовно окремі групи дій будемо нумерувати з метою їх виділення.

Етап 1. Створимо консольний застосунок і наберемо такий текст програми:

```
#include <iostream>

using namespace std;
double a, b, c;

int main()
{
    double x1, x2;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> a;
    d = b * b - 4 * a * a;
    if (d < 0)
        cout << "The equation has no roots\n";
    cout << "   x1 = " << (-b + sqrt(d)) / 2 * a
    cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
}
```

Етап 2. Виконаємо компіляцію. Компілятор виявить 7 синтаксичних помилок і виведе відповідні пояснення у вікно Error List (рис. 2.1).

Програміст повинен шукати синтаксичну помилку в тексті програми перед курсором, причому не обов'язково в тому ж рядку, де розмістився курсор. Якщо виконати подвійний клік мишкою у вікні Error List над будь-яким з рядків, у вікні коду буде здійснений перехід до рядка, в якому вперше знайдена відповідна помилка.

Насправді в даному випадку різних повідомлень тільки чотири:

C2065 'd': undeclared identifier – неоголошений ідентифікатор 'd';

C2146 syntax error: missing ';' before identifier 'cout' – синтаксична помилка: відсутній символ ';' перед ідентифікатором cout;

E0020 identifier 'd' is undefined – невизначений ідентифікатор 'd';

E0065 expected a ';' – очікується ';'.

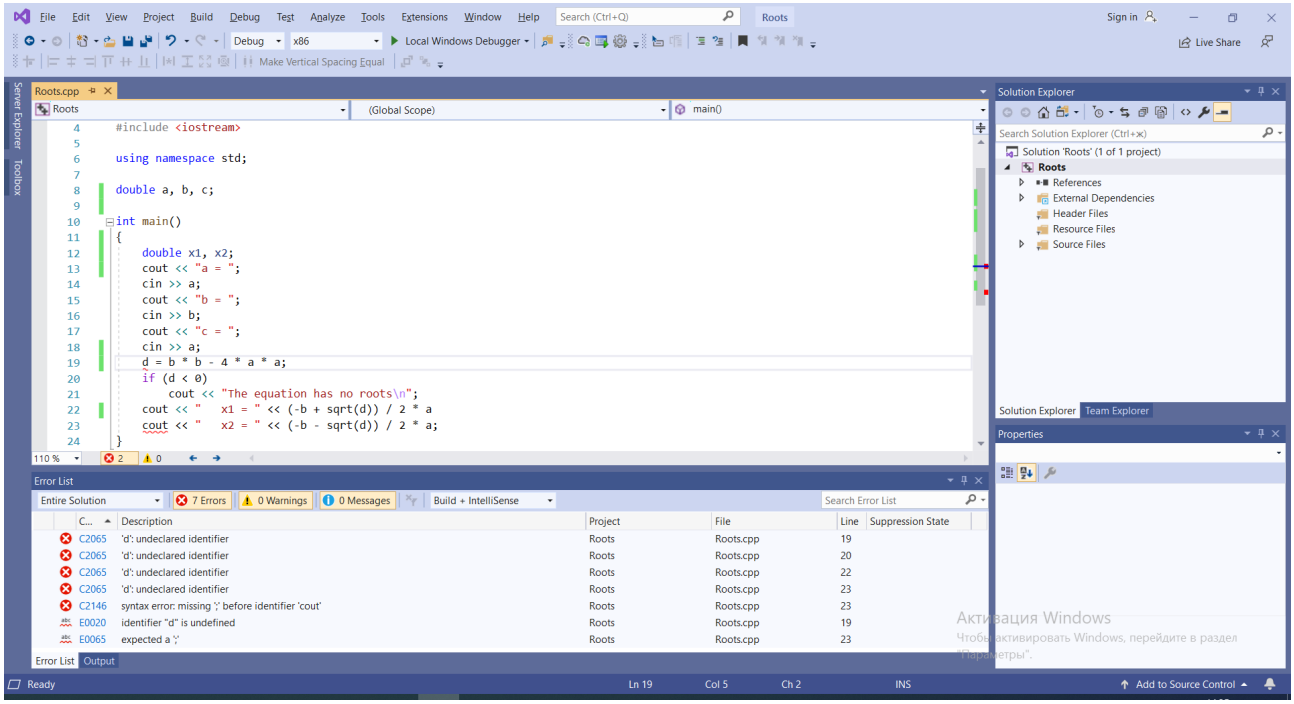


Рис. 2.1. Результати компіляції

Перше з вказаних повідомлень зустрілося 4 рази і говорить про те, що змінна d (для запису значення дискримінанта) використовується, але не визначена в програмі. Ми елементарно забули її оголосити. Усім цим повідомленням відповідає тільки одна помилка, яка зроблена раніше першої точки, де вона була виявлена. Змінна d є допоміжною, тому оголосимо її як локальну змінну всередині функції main. У мові C++ змінні можна оголошувати практично в будь-якому місці програми. Рекомендується робити це безпосередньо перед інструкцією, у якій оголошувана

змінна використовується вперше. Допишемо тип змінної d перед інструкцією для обчислення дискримінанта, сумістивши оголошення змінної з наданням їй значення (ініціалізацією):

```
double d = b * b - 4 * a * a;
```

Помилка C2146 вказує, що інструкції мови C++ повинні закінчуватися символом «крапка з комою». Місце, де виявлена помилка, не співпадає з місцем, де вона була зроблена, – треба дописати символ «крапка з комою» в кінець попереднього рядка, завершивши записану в ньому інструкцію.

Відзначимо, що одна з помилок виправлялася не в тому рядку, де вона була виявлена.

Виправлений текст набирає такий вигляд:

```
#include <iostream>

double a, b, c;

using namespace std;
int main()
{
    double x1, x2;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> a;
    double d = b * b - 4 * a * a;
    if (d < 0)
        cout << "The equation has no roots\n";
    cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
    cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
}
```

Етап 3. Натиснемо інструментальну кнопку 

, запускаючи програму на виконання, і переконаємося, що компіляція і компонування пройдуть успішно і програма почне виконуватися, видавши

запит на уведення коефіцієнту *a*. Якщо увести значення $a = 1$, $b = 2$, $c = 1$, то ми побачимо, що отримані корені $x_1 = -1$, $x_2 = -1$ є правильними. Однак для $a = 1$, $b = 5$, $c = 2$ результати обчислень ($x_1 = -2$, $x_2 = -8$) будуть хибними. Тому треба вже шукати логічні помилки.

Етап 4. Ми можемо підозрювати, що в програмі неправильно обчислюється дискримінант. Помістимо курсор у рядок, наступний за обчисленням значення дискримінанта d, і натиснемо клавішу F5. Після введення вказаного вище другого набору значень коефіцієнтів переконуємося, що дискримінант обчислений невірно (у вікнах Autos і Locals виводиться значення d, яке дорівнює 9 замість 17). Помічаємо, що в тексті програми інструкція

```
d = b * b - 4 * a * a;
```

у якій обчислюється дискримінант квадратного рівняння, є неправильною.

Виправляємо помилку:

```
d = b * b - 4 * a * c;
```

Ми одразу запускали програму на виконання, пропустивши етап компіляції. Про всяк випадок відкомпілюємо код. У результаті будуть отримані два попереджувальні повідомлення:

C4101 'x1': unreferenced local variable – локальна змінна без посилання;

C4101 'x2': unreferenced local variable – локальна змінна без посилання;

Вони свідчать про те, що змінні x1 і x2 ніде в програмі не використовуються. Видалимо їх із коду і спробуємо ще раз виконати програму. Знову отримуємо хибні корені, а саме, $x_1 = 0$, $x_2 = -10$.

Етап 5. Якщо перевірити, як це було виконано на попередньому етапі, обчислене значення дискримінанта, з'ясується, що дискримінант обчислений правильно. Відтак помилка зроблена дещо вище. Натиснемо одну з клавіш F10 або F11 (команди меню Debug ►Step Into або Debug ► Step Over відповідно), почавши тим самим покрокове виконання програми. При цьому занесемо у вікно Watch 1 змінні a, b, c, які будучи глобальними змінними, не відстежуються у вікна Autos і

Locals. Можна побачити, що початкове значення всіх трьох змінних a, b, c дорівнює нулю, а покрокове виконання програми, яке забезпечується натисканням указаних вище клавіш або відповідних інструментальних кнопок, дозволяє відстежити змінення їх значень по ходу введення. Уводячи послідовно значення 1, 5 і 2, побачимо, що змінна a отримала нове значення двічі (кінцеве значення дорівнює 2), а змінна c не вводилася – замість того, щоб увести змінну c, була повторно введена змінна a.

З наявних у програмі двох інструкцій

```
cin >> a;
```

виправимо другу, записавши її так:

```
cin >> c;
```

Якщо тепер запустити програму на виконання, то буде отриманий правильний результат $x_1 = -0.438447$, $x_2 = -0.438447$, який однак ще не свідчить про відсутність логічних помилок, оскільки ми ще не перевіряли випадок відсутності коренів.

Указана помилка була би виявлена раніше, якби змінні a, b, c були визначені як локальні змінні в тілі функції main. У такому разі ще при першій компіляції було б виведено таке попередження:

C6001 Using uninitialized memory 'c' – Використання неініціалізованої пам’яті 'c'.

Оскільки це попередження має відношення до виразу, у якому обчислюється дискримінант, можна було б одразу побачити, що в змінну c введення не здійснюється, і виправити помилку.

Якщо б ми не зреагували на попередження, то після виправлення помилки невизначеності змінної d в результаті компіляції, крім указаного вище попередження, було б виведено таке повідомлення про помилку:

C4700 uninitialized local variable 'c' – неініціалізована локальна змінна 'c'

Оскільки таке повідомлення є не попередженням, а помилкою, ми б були вимушені визначити, чому змінна c не отримала значення.

Етап 6. Запустимо програму на виконання і уведемо тепер такі значення: $a = 1$, $b = 2$, $c = 3$. У цьому разі ми бачимо, що, з одного боку,

програма виводить повідомлення про відсутність коренів, а, з другого боку, вона ще виводить два корені з досить дивними значеннями – nan(ind). Знову треба зайнятися налаштуванням.

Натискаючи декілька разів клавішу F10, бачимо, що інструкції виконуються в потрібній послідовності (по лівій межі вікна коду буде переміщуватися жовта стрілка, вказуючи на рядок, який буде виконуватися наступним), але після виведення повідомлення про відсутність коренів, здійснюється перехід до рядка, в якому обчислюється та виводиться перший корінь, хоч цей рядок не повинен виконуватися. Справа в тому, що в програмі використаний невірний для цієї задачі формат інструкції **if**.

Вставимо перед рядком, де виводиться перший корінь, такий рядок:

```
else
```

Тепер після запуску програми на виконання, ми побачимо, що помилка залишилася, але тільки по відношенню до другого кореня.

За допомогою клавіші F10 переконуємося в тому, що в програмі виконується одна інструкція виведення, виконання якої не повинне було мати місце. Справа в тому, що в цьому випадку дві інструкції

```
cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
```

повинні розглядатися як одна так звана складена інструкція (блок), для чого їх треба забрати у фігурні дужки.

Зробимо це і отримаємо такий текст програми:

```
#include <iostream>

using namespace std;
int main()
{
    double a, b, c, d;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
```

```
    cin >> c;   d = b * b - 4 * a * c;
    if (d < 0)
      cout << "The equation has no roots\n";
    else
    {
      cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
      cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
    }
}
```

Запуск програми зі значеннями $a = 1$, $b = 2$, $c = 3$ тепер приведе до отримання правильного результату.

Етап 7. Запустимо програму на виконання і уведемо нові значення коефіцієнтів: $a = 2$, $b = 5$, $c = 3$. У цьому разі ми бачимо, що обчислені корені (`x1 = -4`, `x2 = -6`) знову є неправильними (повинні бути отримані значення `x1 = -1`, `x2 = -1.5`). Треба знову здійснювати покрокове виконання програми, натискаючи клавішу `F10`. Оскільки ми не передбачили (точніше, знищили) змінні для запису коренів рівняння, прийдеться дивитися на результат, що виводиться на екран користувача. Це можна зробити в даному випадку, але ніяк не може бути рекомендоване для використання. Правильним є або включення у вікно `Watch` виразів для обчислення коренів, або повернення в програму змінних `x1` і `x2` і добавлення інструкцій для запису в ці змінні коренів. У результаті ми побачимо, що всі обчислення до виведення значення першого кореня виконуються правильно, а сам корінь є невірним. Висновок – є помилка в обчисленні кореня. Уважно подивившись на інструкцію, ми бачимо, що в ній порушений порядок виконання операцій, а саме, оскільки операції множення та ділення мають один і той же пріоритет, при обчисленні кореня спочатку здійснюється ділення на 2, після чого отриманий результат помножується на `a`. Щоб отримати правильний результат, необхідно забрати знаменник `2 * a` в дужки, причому це, звісно, потрібно зробити і при обчисленні другого кореня.

Після виправлення тексту отримуємо правильний результат.

Таким чином, одноразове отримання правильного результату не дає гарантії того, що програма буде правильною. Остання помилка не проявлялася раніше, тому що для коефіцієнта a уводилося значення 1, яке

не впливало на результат і в разі попадання цього значення в чисельник, і в разі попадання його в знаменник. Це ще раз говорить про необхідність ретельного підбору тестових даних та багаторазового тестування.

3. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Під час лабораторної роботи потрібно:

1. Створити консольний застосунок і набрати текст програми, наведений на початку розділу 2, і повторити дії, пов’язані з налаштуванням програми.

2. Виконати одне з наведених нижче завдань, пов’язаних з набиранням тексту програми для нового консольного застосунку і збереженням його у файлі.

3. Відредагувати програму і здійснити її налаштування.

4. Виконати дії, пов’язані зі збереженням файлів із новим іменем і завантаженням застосунку з диска.

Завдання 1.

Створіть консольний застосунок, в якому повторіть наведений нижче текст програми. З’ясуйте, що виконує програма.

```
#include <iostream>

using namespace std;
int main()
{
    int a, b, c;

    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    if (a > b)
        cout << "1 - Yes" << endl;
    else
        cout << "1 - No" << endl;
    c = a * b;
    if (c > 100)
        cout << "2 - Yes" << endl;
```

```
    else
        cout << "2 - No" << endl;
    if (a % 2 == b % 2)
        cout << "3 - Yes" << endl;
    else
        cout << "3 - No" << endl;
    cout << "Press any key to close the application." << endl;

    int ch;
    while (!((ch = std::cin.get()) == '\n' || ch == EOF))
        ;
    cin.get();

    return 0;
}
```

Завдання 2.

Нижче наведено код програми з синтаксичними помилками. На основі цього коду створіть консольний застосунок і усуньте в ньому всі знайдені синтаксичні помилки.

```
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
    int a;
    cin >> a;
    cin >> b;
    if (a <> 0 && b <> 0)
    {
        if (a * b > 0)
            cout << 'Yes' << endl
        else
            cout << 'No' << endl
    else
        cout << '?' << endl;
    cout << "Press any key to close the application." << endl;
    _getch();
```

```
    return 0;
}.
```

Завдання 3.

Нижче наведено варіант коду програми, що нормує вектор. У коді містяться помилки. На основі наведеного коду створіть коректно працюючий консольний застосунок.

```
#include <iostream>

using namespace std;
int main()
{
    double a, b, c, d;
    cin >> a >> a;
    c = sqrt(a * a + b * b)
    a = a / d;
    b = b / d;
    cout >> a << "\t" << b << endl;
    cout << "Press any key to close the application." << endl;
    int ch;
    while (!((ch = std::cin.get()) == '\n' || ch == EOF))
        ;
    cin.get();

    return 0;
}
```

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які дії треба виконати для створення нового консольного застосунку?
2. Як створити порожній проєкт для консольного застосунку?
3. Як додати до проєкту порожній cpp-файл?
4. Як добавити до проєкту існуючий модуль (cpp-файл).
5. Як здійснюється збереження вмісту вікна коду?
6. Як здійснюється збереження вмісту вікна коду у файлі з новим іменем?
7. Як відновити раніше закрите вікно коду?

8. Як записати на диск проєкт разом із файлом коду консольного застосунку?
9. Опишіть дії, пов'язані з керуванням курсором у вікні редактора.
10. Як можна здійснювати видалення символів у тексті редакторського вікна? Опишіть дію клавіш видалення.
11. Як здійснюється виділення тексту?
12. Як можна скопіювати фрагмент тексту в інше місце?
13. Як можна перенести фрагмент тексту в інше місце?
14. Яке призначення компілятора?
15. Що являє собою однофайлова програма, написана мовою C++?
16. Чи правда, що компілятор оброблює тільки текст, написаний програмістом?
17. Як поділяються помилки в програмі? Опишіть їх особливості.
18. Що створює компілятор у разі відсутності синтаксичних помилок?
19. Яке призначення компонувальника (редактора зв'язків) і що є результатом його роботи?
20. Яке призначення препроцесорних директив і як вони записуються?
21. У чому відмінність команд Compile, Build і Build Solution?
22. Де і як виводяться результати роботи компілятора і компонувальника?
23. Яке призначення діалогового віконця помилки компонування? Як можна заборонити/відновити його виведення?
24. Що відслідковується при покроковому виконанні програми?
25. У чому полягає відмінність режимів покрокового виконання Step Into та Step Over?
26. Що таке «виконання до курсору» і як воно забезпечується?
27. Що таке точка переривання і як вона встановлюється та видаляється?
28. У чому відмінність точки переривання і виконання до курсору?
29. Чи можна одночасно встановити декілька точок переривання?
30. Для чого використовуються вікна Autos, Locals і Watch? У чому їх відмінність?
31. Як внести нове ім'я у вікно Watch?
32. Як видалити ім'я з вікна Watch?

33. Чи можна в процесі покрокового виконання програми здійснювати примусову модифікацію значень змінних? У разі позитивної відповіді дайте пояснення.
34. Чи є можливість отримання інформації про значення виразів у процесі покрокового виконання? Дайте пояснення при позитивній відповіді.

СПИСОК ЛІТЕРАТУРИ

1. Visual C++ .NET. Классика программирования / под ред. О. Е. Степаненко. – Москва : Научная книга; Киев : Букинист, 2010. – 768 с.
2. Страуструп, Б. Язык программирования Си++. / Б. Страуструп. – Москва : Бином, 1999. – 991 с.
3. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – Москва : Финансы и статистика, 1992. – 272 с.
4. Либерти, Джесс. Освой самостоятельно C++ за 21 день : учеб. пособ. / Джесс Либерти. – Москва : Вильямс, 2001. – 816 с.
5. Либерти, Дж. Освой самостоятельно C++ за 21 день : учеб. пособ. / Джесс Либерти. – Москва : Вильямс, 2001. – 816 с.
6. Савитч, У. Язык C++. Курс объектно-ориентированного программирования / Уолтер Савитч. – Москва : Вильямс, 2001. – 704 с.
7. Шилдт, Г. C++: руководство для начинающих / Герберт Шилдт. – Москва : Вильямс, 2005. – 672 с.
8. Шилдт, Г. Самоучитель C++ / Г. Шилдт. – Санкт-Петербург : BHV-Петербург, 2003. – 688 с.
9. Шилдт, Г. Полный справочник по C++ / Герберт Шилдт. – Москва : Вильямс, 2006. – 800 с.
10. Шпак, З. Я. Програмування мовою С / З. Я. Шпак. – Львів : Оріяна-Нова, 2006. – 432 с.
11. Вінник, В. Ю. Алгоритмічні мови та основи програмування: мова Сі / В. Ю. Вінник. – Житомир: ЖДТУ, 2007. – 328 с.
12. Гнатів, Б. В. Програмування на С(С++). Парадигма процедурного програмування : навч. посіб. / Б. В. Гнатів, Л. Б. Гнатів. – Львів : Растр-7, 2017. – 264 с.
13. Шпак, З. Я. Програмування мовою С : навч. посіб. / Зореслава Ярославівна Шпак. – Львів : Оріяна-Нова, 2006. – 431 с.

14. Кравець, П. О. Об’єктно-орієнтоване програмування : навч.посібн. / П. О. Кравець. – Львів : Львівська політехніка, 2012. – 624 с.
15. Креневич, А. П. С у задачах і прикладах : навч. посіб. / А. П. Креневич, О. В. Обвінцев. – Київ : Київський університет, 2011. – 208 с.
16. Безменов, М. І. Збірник задач із програмування : збірник / М. І. Безменов. – Харків : НТУ «ХПІ», 2014. – 304 с.

НАВЧАЛЬНЕ ВИДАННЯ

Методичні вказівки
до виконання лабораторної роботи
«Створення консольного застосунку мовою C++
у Microsoft Visual Studio 2019» з курсу «Алгоритмізація
та програмування» для студентів спеціальності 124 «Системний
аналіз» і курсу «Інформатика і програмування» для студентів
спеціальності 186 «Видавництво і поліграфія»

Укладач:
БЕЗМЕНОВ Микола Іванович

Відповідальний за випуск Ю. І. Дорофєєв
Роботу до видання рекомендував І. П. Гамаюн
Комп'ютерна верстка М. І. Безменов

У авторській редакції

План 2022 р., поз. 297

Підп. до друку 15.02.2023 р. Формат 60×84 1/16. Папір офсетний.
Друк – цифровий. Гарнітура Таймс. Ум. друк. арк. 1,78.
Наклад 50 пр. Зам. № 43. Ціна договірна

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК№ 3657 від 27.12.2009 р.
61002, Харків, вул. Фрунзе, 21

Друкарня «ФОП Пісня О. В.»
Свідоцтво про державну реєстрацію ВО2 № 248750 від 13.09.2007 р.
61002, Харків, вул. Гіршмана, 16а, кв. 21, тел. (057) 764-20-28