ПРАКТИЧНА РОБОТА

ПРОЦЕС РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ С В ІНТЕГРОВАНОМУ СЕРЕДОВИЩІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ CODE::BLOCKS

1.1. Мета роботи

Ознайомитися з основними етапами створення програм. Ознайомитися з основними особливостями мови програмування на мові С. Ознайомитися з основними особливостями інтегрованого середовища розробки програмного забезпечення Code::Blocks. Навчитися складати, компілювати, компонувати та виконувати консольні програми.

1.2. Теоретичні відомості

1.2.1.Розроблення програм на мові С

Щоб отримати загальне уявлення про програмування, розділимо процедуру написання програми на сім основних етапів (рис. 1.1) [6]. Зауважимо, що це ідеалізація. На практиці, особливо в разі великих проектів, з'явиться необхідність переміщатися назад і вперед, використовуючи те, що отримано на більш пізньому етапі, для уточнення результатів, отриманих на більш ранній стадії.

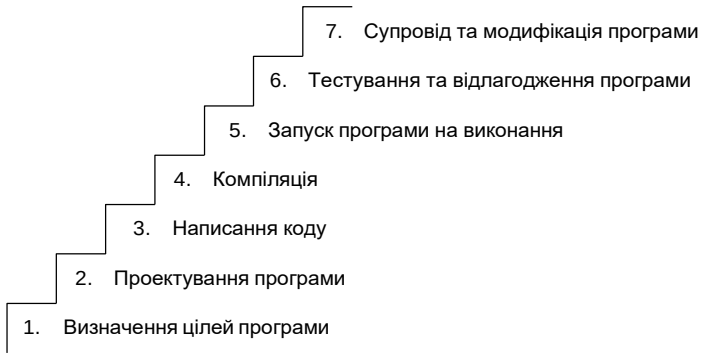

Рис. 1.1 Сім основних етапів програмування

Етап 1: Визначення цілей програми

Цілком природно, необхідно починати з чіткого бачення того, що програма повинна робити. Необхідно усвідомити якими є вхідні дані, які операції та маніпуляції необхідно над ними провести та якими будуть вихідні дані програми. На цьому етапі необхідно мислити в загальних термінах, а не в термінах якоїсь конкретної мови програмування.

Етап 2: Проектування програми

Після того, як стане зрозумілою концептуальна картина того, що програма повинна зробити, необхідно вирішити, як вона повинна це зробити. Яким

повинен бути користувальницький інтерфейс? Як повинна бути організована ця програма? Якими будуть потенційні користувачі? Скільки часу буде потрібно для завершення розроблення програми?

Необхідно вирішити, як представляти дані у програмі і, можливо, у допоміжних файлах, а також які методи слід використовувати для обробки даних. На початковому етапі вивчення програмування відповіді на ці питання не викличуть труднощів, але коли ситуація стане більш складною, то прийде розуміння, що ці рішення вимагатимуть врахування великої кількості обставин. Правильний вибір способу подання інформації може істотно полегшити розробку програми і обробку даних.

Підкреслимо ще раз, що на цьому етапі необхідно мислити загальними категоріями і не думати про конкретний програмний код, але деякі з рішень можуть бути засновані на загальних характеристиках мови програмування.

Етап 3: Написання коду

Тепер, коли проект програми створений, можна приступати до її реалізації, для чого необхідно написати програмний код. Саме на цій стадії будуть потрібні всі знання мови програмування. Динаміка процесу залежить від середовища програмування, яке використовується.

До числа робіт, що потрібно зробити на цьому етапі, відноситься документування виконаних дій. Найпростішим способом документування є коментування зі зрозумілими поясненнями, якими забезпечується програмний код.

Етап 4: Компіляція

Базова стратегія програмування полягає в тому, щоб використовувати програми, які перетворять вихідний код на мові програмування у виконуваний файл, який містить готовий до виконання програмний код на машинній мові. Ця робота виконується в два етапи: компіляція і компонування.

Компілятор – це програма, в обов'язки якої входить перетворення вихідного коду у виконуваний код, тобто компіляцію. Компілятор також перевіряє, чи не містить помилок програма. Коли компілятор знаходить помилки, він повідомляє про їх наявність і не створює виконуваний файл. Результатом роботи компілятора є проміжний код.

Компонувальник – програма, що комбінує цей проміжний код з іншими необхідними проміжними кодами, в результаті виходить виконуваний файл, що містить виконуваний код.

Виконуваний код – це команди на власній машинній мові конкретного комп’ютера, що безпосередньо ним виконуються.

Проміжний код зазвичай представляється у вигляді *об’єктного коду* – фактично виконуваного коду, який однак не може бути виконаний на конкретному комп’ютері, оскільки в ньому відсутні деякі важливі пункти.

Перший елемент, якого бракує в файлі об'єктного коду це код запуску, що представляє собою код, який діє в якості інтерфейсу між програмою і операційною системою. Наприклад, можна запускати програму на однакових комп'ютерах, один з яких функціонує під управлінням Windows, а інший – під управлінням Linux. В обох випадках обладнання одне і те ж, так що використовується один і той ж об'єктний код, але при цьому потрібні різні коди запуску для Windows і для Linux, оскільки ці системи підтримують програми по-різному. Другим відсутнім елементом є коди бібліотечних програм. Практично всі С-програми використовують стандартні бібліотечні функції. Об'єктний файл не містить код самих функцій, він просто містить команду, що вимагає використання цих функцій. Фактичний код зберігається в файлі, що отримав назву бібліотеки.

Роль компонувальника полягає в тому, щоб зібрати воєдино ці три елементи – об'єктний код, стандартний код запуску і бібліотечний код – з подальшим запам'ятовуванням в окремому файлі, який називається виконуваним. Що стосується бібліотечного коду, то компонувальник витягує тільки код, необхідний для функцій, що викликаються з бібліотеки (рис. 1.2).

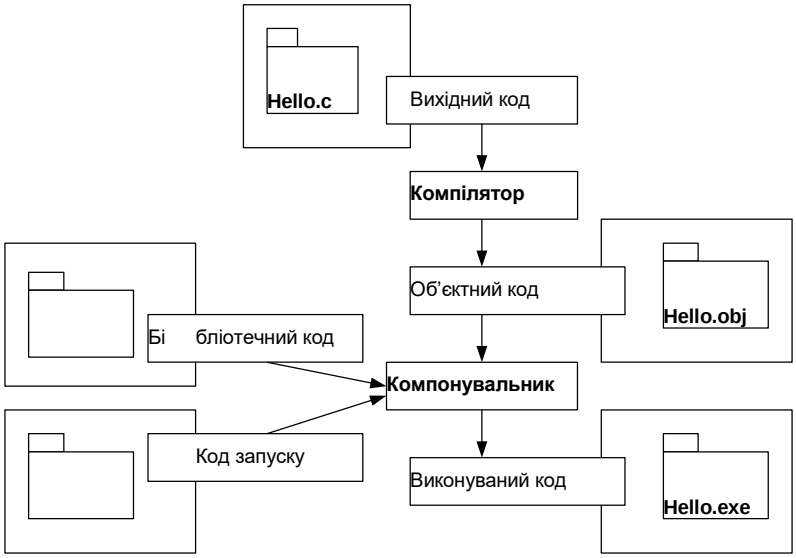

Рис. 1.2 Компілятор і компонувальник

В С використовується такий двоетапний підхід для модульної організації програм. З’являється можливість компілювати індивідуальні модулі окремо, а потім за допомогою компонувальника об'єднати скомпільовані модулі. Таким чином, якщо буде потрібно змінити якийсь один модуль, не потрібно буде повторно компілювати інші модулі. Крім того, компонувальник зв’язує програму з попередньо відкомпільованим бібліотечним кодом.

Етап 5: Запуск програми на виконання

Як правило, виконуваний файл є програмою, яку можна запускати на виконання. Щоб запустити програму, в багатьох відомих середовищах, включаючи консолі Windows, Unix, Linux, необхідно ввести з клавіатури ім'я виконуваного файлу. Інші середовища можуть зажадати введення команди запуску або використання будь-якого іншого механізму.

Середовища IDE (Integrated developtent environtents – *інтегроване середовище розробки,* ICP), подібні до тих, що поставляються для Windows і Macintosh, дають змогу редагувати і виконувати програми на С всередині середовища, вибираючи відповідні пункти меню або натискаючи спеціальні клавіші. Отримана програма може бути запущена на виконання безпосередньо з операційної системи шляхом одиночного або подвійного клацання на імені файлу або на відповідній піктограмі.

Етап 6: Тестування і налагодження програми

Той факт, що програма працює – хороший знак! Тим не менш, є ймовірність, що вона не працює належним чином. Звідси випливає, що необхідно переконатися, що програма робить саме те, що і повинна. Досить часто в програмах будуть виявлятися помилки.

Відлагодження – це процес виявлення і виправлення програмних помилок. Допущення помилок є природною складовою процесу навчання. Вони притаманні програмуванню. Існує багато можливостей зробити помилку. Можна зробити принципову помилку в проекті програми. Можна некоректно реалізувати хорошу ідею. Можна випустити з уваги неприпустимі вхідні дані. Можна неправильно використовувати конструкції самої мови програмування. Можна допускати помилки при наборі коду з клавіатури. Можна неправильно розставити дужки і так далі.

На щастя, ситуація небезнадійна, хоча можуть настати такі моменти, коли здасться, що це саме так. Компілятор відстежує багато видів помилок, крім того, можна зробити певні зусилля, щоб допомогти самому собі в пошуку помилок, що не відловив компілятор.

Інтегровані середовища розробки програмного забезпечення зазвичай містять засоби для покрокового виконання написаних програм з можливістю перегляду поточних значень змінних, тощо. Це дає змогу перевірити хід виконання програми і зрозуміти, що в ній відбувається не так, як задумано.

Етап 7: Супровід і модифікація програми

Коли програма створюється для себе або для кого-небудь ще, то, швидше за все, припускається, що вона буде використовуватися досить часто. Якщо це так, можливо з'являться причини для внесення в неї змін. Можливо існує якийсь незначний дефект, який проявляється при введенні імені, що починається з букв "Zz", або виникає бажання поліпшити щось в програмі. Або з часом необхідно

добавити в неї нову функціональну можливість. Можна адаптувати програму для виконання в різних комп'ютерних системах. Рішення задач подібного роду істотно спрощується, якщо програма була чітко документована а її код був написаний у відповідності до перевірених на практиці рекомендацій.

Програмування зазвичай не є таким послідовним процесом, яким є описаний вище процес. Час від часу прийдеться переміщатися туди і назад по етапах. Наприклад, коли пишеться програмний код, можна прийти до висновку, що обраний раніше план нездійсненний. Можна побачити кращий спосіб вирішення завдання. Можливо, після аналізу виконання програми виникне бажання змінити проектне рішення. Документування виконуваних дій допоможе переміщатися по етапах туди і назад.

Багато з тих, хто вивчає програмування нехтують етапами 1 і 2 (визначення цілей і побудова проекту програми) та переходять безпосередньо до етапу 3 (Написання програми). Перші написані програми будуть досить простими, щоб "прокрутити" весь процес розробки в голові. Якщо допускається помилка, її легко знайти. У міру того як програми стають все більшими і складнішими, уявне представлення програми починає підводити, а на виявлення помилок йде все більше часу. В кінцевому підсумку ті, хто знехтував стадією планування, приречені на марну витрату часу, на довгі години плутанини, до того ж отримуючи потворні, такі, що погано функціонують і важкі для розуміння програми. Чим більше і складніше завдання, тим більше часу доводиться витрачати на планування його рішення. Висновок, який випливає з усього вищесказаного, полягає в тому, що необхідно виробити в собі звичку складати плани, перш ніж приступати до написання власне коду.

1.2.2.Мова програмування С

С [7] – процедурна мова програмування загального призначення, розроблена у 1972 році у Bell Telephone Laboratories з метою написання нею операційної системи UNIX.

Процедурне програмування – це парадигма програмування, заснована на концепції виклику процедури. Процедури, також відомі як підпрограми, методи або функції. Процедури містять певну послідовність кроків до виконання. В ході виконання програми будь-яка процедура може бути викликана з будь-якого місця програми.

Протягом останніх десятиліть С стала однією з основних і найбільш широко поширених мов програмування. І хоча багато програмістів перейшли на більш претензійну об’єктно-орієнтовану мову С++, але мова С сама по собі все ще залишається важливою, особливо на шляху вивчення та переходу до С++.

У міру вивчення С можна переконатися, що вона має багато переваг, зокрема:

- компактний програмний код, відносно невеликі програми;
- кросплатформовість;
- швидкодія;
- наявність потужних структур управління;
- велика кількість прикладних програмних бібліотек найрізноманітнішого призначення.

Мова програмування С відрізняється мінімалізмом. Автори мови хотіли, щоб програми на ній легко переводилися в машинний код, щоб кожній елементарній складової програми після цього відповідало досить невелике число машинних команд. С створювалася з однією важливою метою: зробити простішим написання великих програм з мінімумом помилок за правилами процедурного програмування, не додаючи до коду програм нічого зайвого. Тому, С має наступні важливі особливості:

- просту мовну базу, з якої винесена в окремі бібліотеки велика кількість функціональних можливостей, наприклад математичні функції або функції управління файлами;
- орієнтацію на процедурне програмування;
- строгу систему типів змінних програм, що уберігає від безглуздих операцій;
- використання, так званого, препроцесора для, наприклад, визначення макросів і включення файлів з вихідним кодом;
- безпосередній доступ до пам'яті комп'ютера через використання вказівників на ділянки оперативної пам’яті;
- мінімальне число ключових слів;
- структури і об'єднання – визначені користувачем об’єднані типи даних, якими можна маніпулювати як одним цілим.

Мова С сильно вплинула на історичний розвиток мов програмування, що призвело до появи цілого класу, так званих, С-подібних мов.

Не зважаючи на те, що мова С розроблялася як мова "високого рівня" програмування, тобто така, що орієнтована в першу чергу на програмістів та дає їм гнучкі можливості доступу та маніпуляції ресурсами комп’ютера, за сучасними мірками вона є порівняно "низького рівня". У ній **немає**:

- прямих операцій над такими об’єктами як множини, стрічки, списки і масиви;
- операцій які маніпулюють цілими масивами або стрічками, натомість використовуються структури;
- засобів розподілу пам’яті окрім можливості визначення статичних змінних і стекового механізму при виділенні місця для локальних змінних функцій;

- засобів вводу-виводу і методів доступу до файлів.

Все це механізми високого рівня, які в мові С реалізуються за допомогою окремих бібліотечних функцій.

1.2.3.Інтегроване середовище розробки Code::Blocks

Інтегроване середовище розробки (ICP, англ. Integrated development environment, IDE) – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду. Деякі середовища розробки містять у базовому комплекті компілятор та компонувальник. Деякі інтегровані середовища розробки містять систему керування версіями або інструменти для полегшення розробки графічного інтерфейсу користувача.

Code::Blocks – вільне кросплатформове середовище розробки програмного забезпечення [8], доступне за посиланням http://www.codeblocks.org/.

Існує також портативна версія з вбудованим компілятором MinGW (популярним варіантом GNU GCC Compiler для Windows), яка не потребує інсталювання на комп'ютері. Доступна за посиланням http://codeblocks.codecutter.org/.

Розглянемо алгоритм роботи з ICP[1]:

Перший запуск Code::Blocks

Середовище запускається з допомогою exe-файлу **CbLauncher.exe**. Якщо інсталювання пройшло без помилок, то після запуску Code::Blocks, під час його завантаження з'явиться заставка, на якій буде відображена версія. На рис. 1.3 наведено зображення заставки Code::Blocks де вказано номер збірки (версії).

При першому запуску з'явиться діалогове вікно зі списком знайдених компіляторів, з них треба вибрати той компілятор, який буде використовуватися за замовчуванням. Для виконання практичних робіт достатньо вибрати пропонований за замовчуванням GNU GCC Compiler і натиснути ***OK***[2].

Рис. 1.3 Заставка Code::Blocks

[1] На прикладі лінійки операційних систем Windows.

[2] Вказаний компілятор не є частиною ICP Code::Blocks і може інсталюватися на комп'ютері окремо. Доступний за посиланням http://www.mingw.org/.

На рис. 1.4 представлено діалогове вікно "Порада дня". Якщо необхідно, щоб воно не з'являлося при наступному запуску Code::Blocks, достатньо зняти галочку "Show tips at startup".

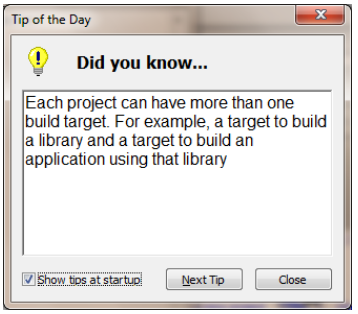

Рис. 1.4 Вікно "Порада дня"

Далі Code::Blocks запропонує використовувати себе в якості програми за замовчуванням для всіх файлів з розширенням **`.c`** / **`.cpp`** і **`.h`** / **`.hpp`**, які є на комп'ютері. Після вибору необхідної дії та натискання ***OK***, з'явиться головне вікно програми, представлене на рис. 1.5. Все, попереднє налаштування закінчено, можна приступати до роботи в Code::Blocks.

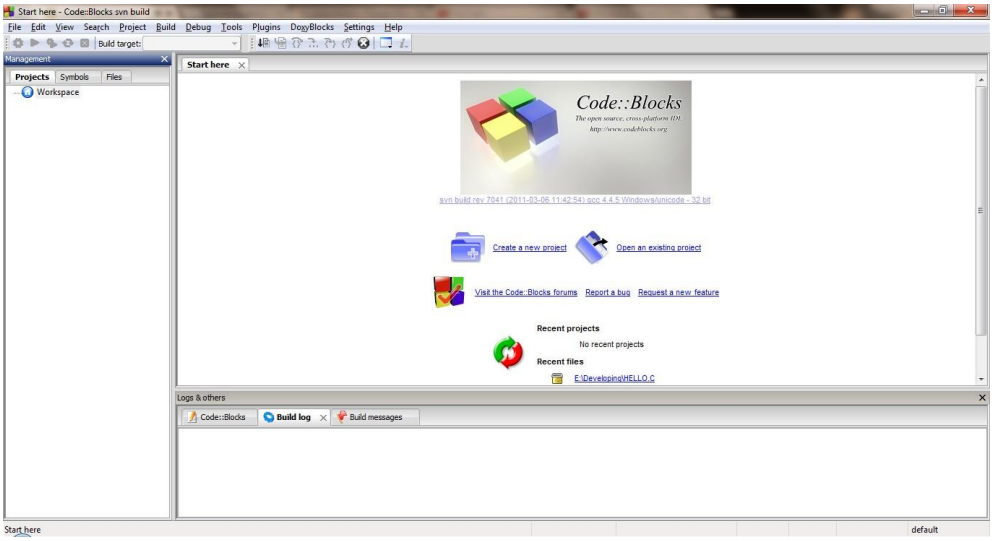

Рис. 1.5 Основне вікно Code::Blocks

Створення проекту в Code::Blocks на мові С

Під *проектом* розуміється деяка логічна організація файлів, необхідних для створення конкретного програмного продукту. Наприклад це можуть бути коди програм, піктограми з зображеннями іконок інтерфейсу користувача, файли налаштування середовища з спеціальними інструкціями для компілятора та

компонувальника, об'єктні файли, виконувані файли тощо. Середовище розробки програмного забезпечення організовує всі ці файли в окремі проекти.
На рис. 1.6 можна побачити вікно програми зі сторінкою "швидкого старту", що відкривається за замовчуванням при кожному запуску. ***Create a new project*** запускає майстер "створення проектів". ***Open an existing project***, відкриває діалогове вікно ***Open file***, в якому можна відкрити файл, вже існуючого проекту. Це ж діалогове вікно ***Open file*** можна викликати натиснувши поєднання клавіш ***Ctrl+O***. У списку ***Recent projects*** представлений список проектів, які були відкриті недавно. У списку ***Recent files***, представлений список файлів, що відкривалися недавно поза проектом.

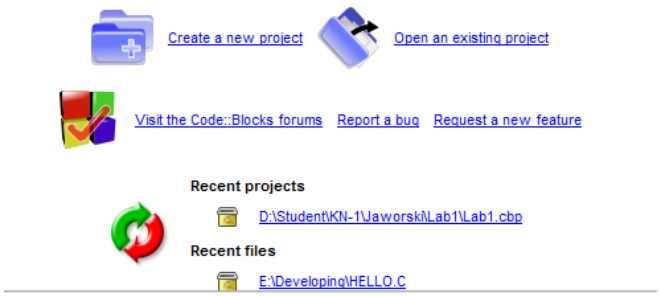

Рис. 1.6 Швидкий старт

Для створення проекту треба вибрати пункт меню ***File→New→Project ...*** або у вікні "швидкого старту" пункт ***Create a new project***, як показано на рис. 1.7.

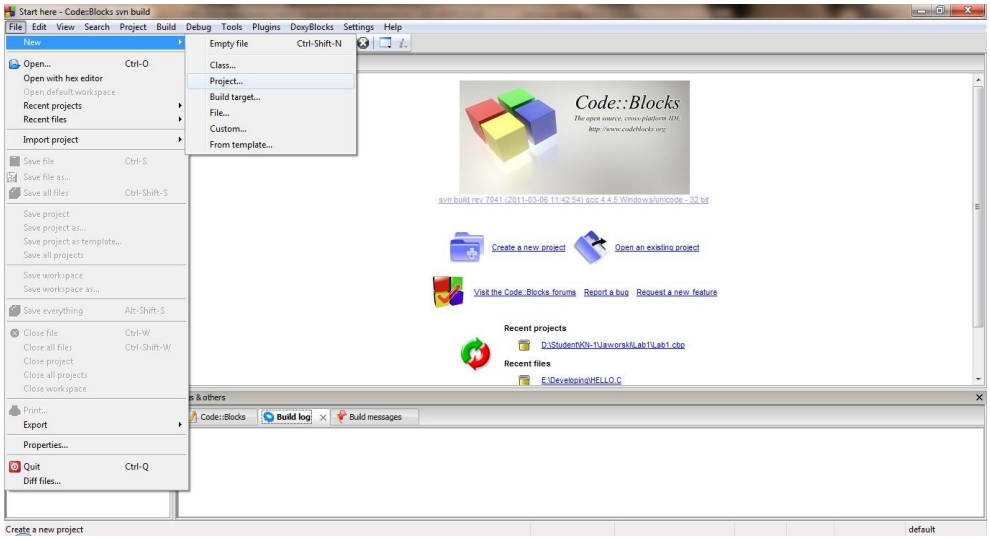

*Рис. 1.7 Загальний вид ICP і відкритого меню **File→New***

Далі з'явиться діалогове вікно ***New from template***, в якому треба вказати шаблонпроекту. Для виконання практичних робіт треба вибрати ***Console application*** (рис. 1.8). І натиснути ***GO***, що викличе вікно майстра створення консольних програм.

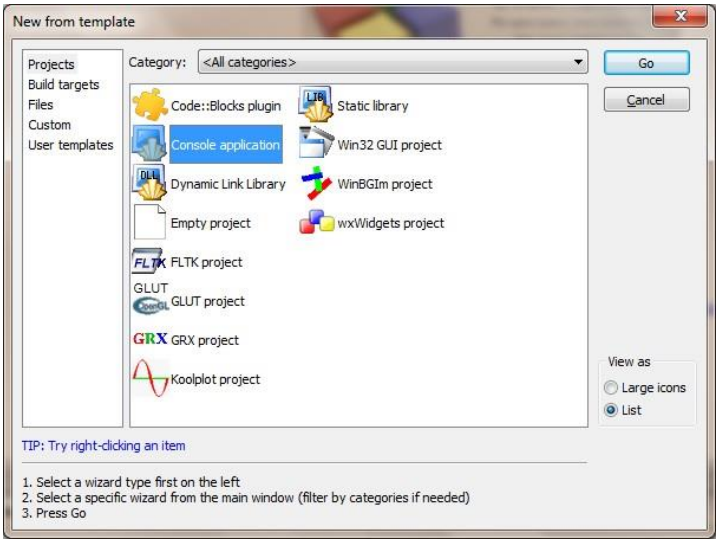

Рис. 1.8 Діалогове вікно вибору шаблону проекту ***New from template*** *додатків*

З'явиться діалогове вікно майстра з першою сторінкою рис. 1.9, де буде виведено привітання, і можливість поставити галочку ***Skip this page next time***, якщо потрібно, щоб наступного разу при створенні нового проекту ця сторінка майстра пропускалася. Для продовження слід натиснути ***Next***.

Рис. 1.9 Діалогове вікно майстра з першою сторінкою "Ласкаво просимо в майстер нового консольного додатку"

Відкриється сторінка (рис. 1.10), в якій потрібно вибрати якою мовою буде написана програма. Тут необхідно вибрати мову C та натиснути ***Next*** для переходу до наступної сторінки майстра рис. 1.11.

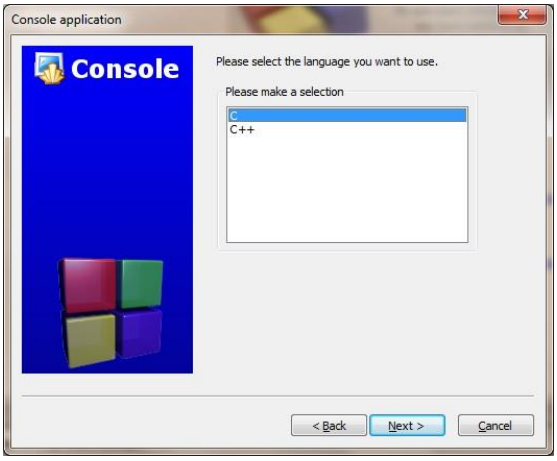

Рис. 1.10 Сторінка майстра, де потрібно вибрати мову програмування

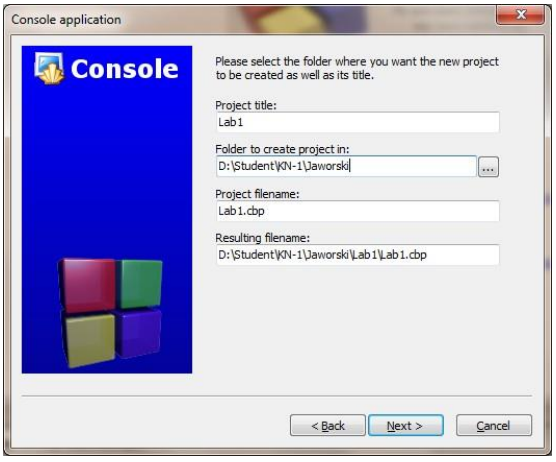

Рис. 1.11 Сторінка майстра, де треба вказати ім'я створюваного проекту і каталог, де він буде розміщений

Тут слід ввести в поле ***Project title*** ім'я проекту, в поле ***Folder to create project in*** вказати каталог, в якому буде зберігатися проект. Якщо проект створюється вперше, то треба створити каталог, де він і майбутні проекти будуть зберігатися. Для цього треба натиснути на кнопку з трикрапкою. Після цього відкриється вікно огляду катологів. Якщо вже існує каталог, де зберігаються практичні роботи, слід його вибрати, в іншому випадку треба його створити. Третє і четверте поля можна не заповнювати. Вказати ім'я файлу проекту можна в поле ***Project filename***. В поле ***Resulting filename*** вказується

кінцеве ім'я каталогу і проекту, отримане при заповненні попередніх полів. Після заповнення всіх необхідних полів слід натисути ***Next***.

Відкриється наступна сторінка (рис. 1.12). Тут треба вибрати компілятор, який буде використовуватися для компіляції програми (поле ***Compiler***) а також сценарії збірки. Сценарії допомагають отримати кілька версій однієї програми. Пропонуються два сценарії за замовчуванням, це ***Debug*** і ***Release***.

Debug – сценарій компіляції, який використовується при відлагодженні програми. У цьому випадку компілятор додасть у виконуваний код програми спеціальні ділянки, що будуть використовуватися для покрокового виконання та можливості перегляду поточного значення змінних, тощо. Зазвичай це унеможливлює використання виконуваного модуля програми на інших комп'ютерах, тому слід розглядати цей варіант як чорновик.

Release – сценарій компіляції для кінцевого результату програми, який можна передавати комусь у користування.

Якщо не потрібно створювати якийсь із цих сценаріїв, слід зняти відповідну галочку. Хоча б один з сценаріїв повинен бути обов'язково зазначений!

У кожному із сценаріїв пропонуються каталоги, куди будуть поміщатися файли, скомпільованої програми: ***Output dir*** – для всіх файлів, ***Objects output dir***– тільки для об'єктних файлів. У звичайних випадках ці каталоги міняти не потрібно. Після натискання на кнопку ***Finish*** проект буде автоматичний створений і відкритий[1].

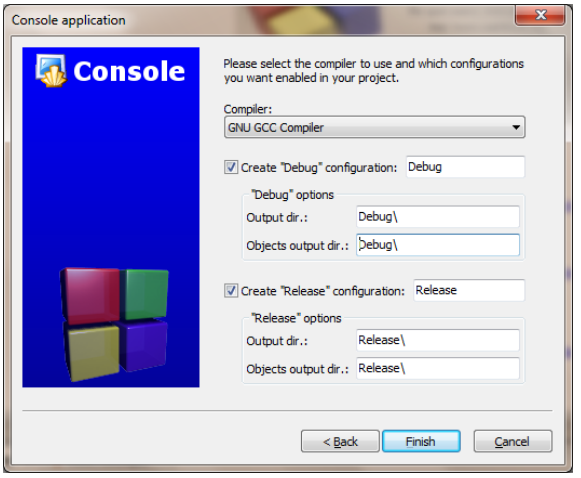

Рис. 1.12 Сторінка майстра, де потрібно вибрати компілятор і визначити можливі сценарії збірки

[1] В принципі, для маленьких програм створювати проект не обов'язково, однак це значно спрощує роботу. Можливості відлагодження доступні тільки при створенні проекту.

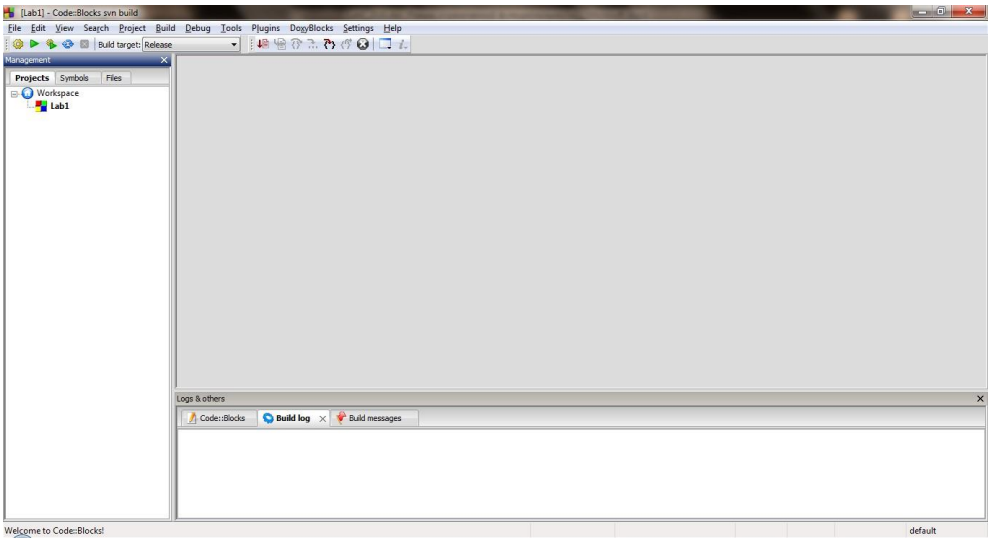

Рис. 1.13 Головне вікно Code::Blocks

Перші кроки в Code::Blocks після створення проекту

Після того, як було створено проект практичної роботи, з'явиться головне вікно з закритим редактором коду (рис. 1.13), де у вигляді дерева відображена ієрархічна структура проекту. Проект належить робочому простору ***Workspace***. Проект носить те ім'я, яке йому дали при створенні.

Необхідно створити файл з кодом програми, для цього слід виконати команду ***File→New→File ...***, після чого відкриється вікно створення файлів за шаблоном (рис. 1.14), де необхідно вибрати ***C/C++ source*** і натиснути ***GO***. З'явиться діалогове вікно майстра аналогічне до попереднього випадку.

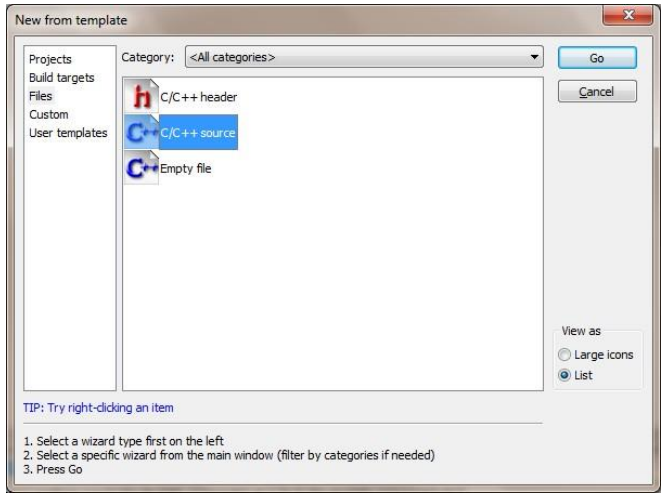

Рис. 1.14 Діалогове вікно вибору шаблону файлу ***New from template***

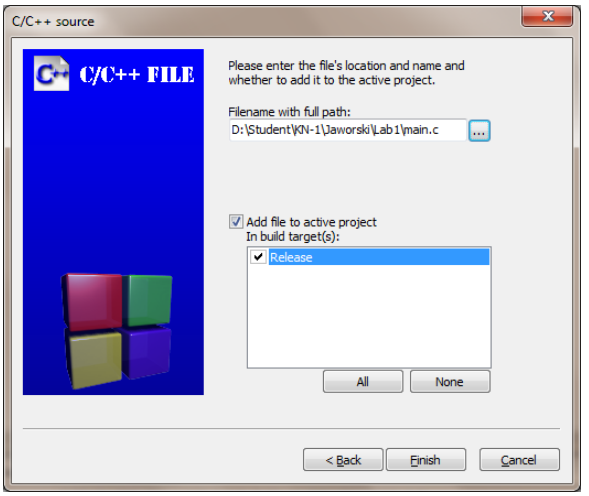

Рис. 1.15 Сторінка майстра, де треба вказати ім'я створюваного файлу та приєднати його до проекту

Виконуючи підказки майстра, необхідно вказати ім'я файлу та приєднати його до активного проекту (рис. 1.15). Після натискання кнопки ***Finish*** файл відкриється для редагування. Можна побачити, що дерево проекту у панелі ***Management*** також змінилося (рис. 1.16).

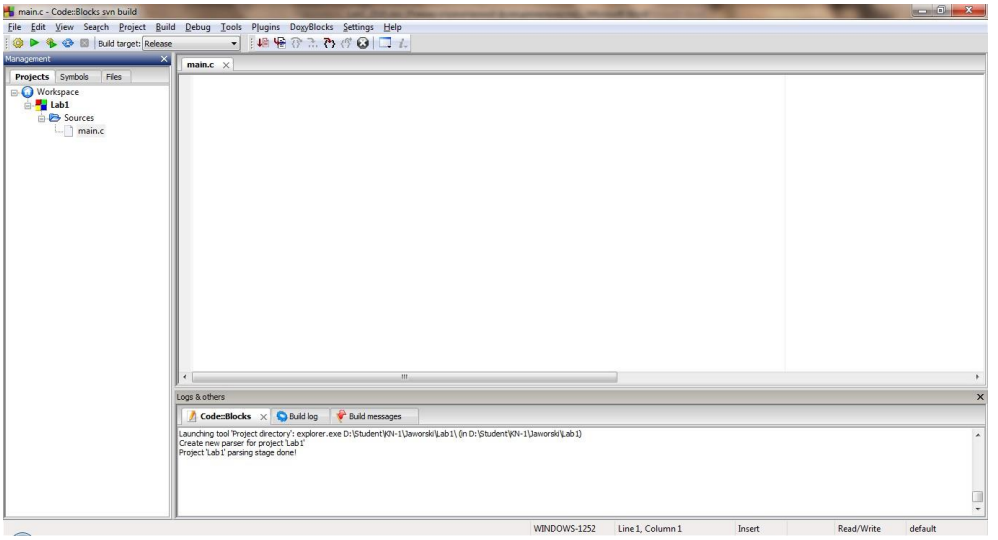

Рис. 1.16 Головне вікно Code::Blocks з відкритим для редагування файлом

На рис. 1.16, можна побачити панель, де відображаються заголовки всіх відкритих файлів. Якщо потрібно закрити якийсь файл, то слід натиснути на хрестик поруч з ім'ям цього файлу. Для закриття файлу, що відображається в даний момент, також можна скористатися поєднанням клавіш ***Ctrl+W.*** Для

переходу між відкритими файлами можна скористатися поєднанням клавіш ***Ctrl+Tab,*** або натискаючи на їх заголовки мишею. Якщо файл був змінений, то на його вкладці зліва від імені файлу з'являється зірочка. Для збереження файлу треба натиснути клавіші ***Ctrl+S,*** або мишкою в панелі інструментів натиснути на кнопку ***Save.***

У нижній частині зліва, в рядку стану, відображається шлях і ім'я відкритого в даний момент файлу.

За допомогою меню ***View*** (рис. 1.17) можна керувати зовнішнім виглядом Code::Blocks. Щоб відобразити або сховати панелі інструментів, треба зайти в меню ***View→Toolbars*** і відзначити відповідні панелі:

- ***Main*** – головна панель інструментів, на неї виводяться основні дії по роботі з проектами;
- ***Code completion*** – панель дозволяє переглядати об'єкти коду;
- ***Compiler*** – панель, на якій знаходяться кнопки управління компіляцією програми;
- ***Debugger*** – панель, на якій знаходяться кнопки управління відлагодженням програми.

Пункт меню ***View→Logs*** або клавіша ***F2*** відображають або ховають вікно повідомлень компілятора ***Logs*** внизу екрану.

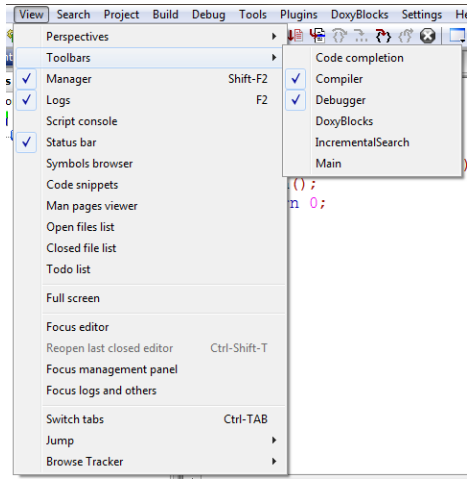

Рис. 1.17 Меню ***View****, де можна налаштувати зовнішній вигляд Code::Blocks*

Робота з більш, ніж одним відкритим проектом

Якщо відкрито одночасно декілька проектів (рис. 1.18), то той проект, який виділений жирним шрифтом, є активним проектом і незалежно від того, який файл відкритий в основному вікні в даний момент, компілюватиметься завжди буде саме активний проект. На малюнку він додатково виділений червоним кольором.

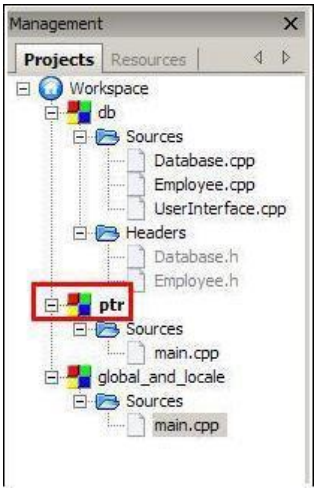

Рис. 1.18 Панель ***Management*** *с кількома відкритими проектами*

Для перемикання проектів слід навести курсор на той проект, який необхідно зробити активним і натисніть по ньому правою кнопкою миші. З'явитися меню, що випадає (рис. 1.19). У ньому слід вибрати пункт ***Activate project.*** Коли проект необхідно закрити, то слід вибрати пункт меню ***Close project.***

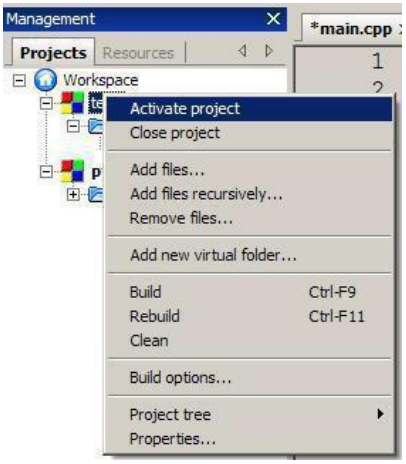

Рис. 1.19 Меню проекту

Компіляція та компонування проекту (практичної роботи)

Для компонування проекту, компіляції та запуску програми, треба натиснути клавішу ***F9*** або пункт в меню ***Build→Build and run*** (рис. 1.20). Якщо потрібно тільки скомпілювати проект без запуску, то слід вибрати пункт меню ***Build*** або натисніть клавіші ***Ctrl+F9.*** Якщо потрібно видалити тимчасові файли проекту, то треба вибрати пункт меню ***Clean.***

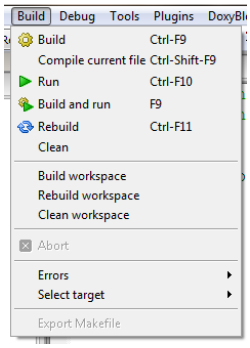

Рис. 1.20 Меню ***Build***

При створенні проекту було потрібно визначити сценарії збірки програми. Два сценарії збірки ***Debug*** або ***Release*** дають змогу отримати також два незалежних варіанти програми з різними опціями складання, і відповідно з різними опціями оптимізують додаток. Для того, щоб ними скористатися треба вибрати відповідний сценарій збірки. Це можна зробити в панелі інструментів ***Compiler***, де можна задати один з двох сценаріїв зі списку ***Build target.***

Змінити опції компіляції для кожного із сценаріїв можна у вікні ***Project build options*** (рис. 1.21), вибравши відповідний сценарій ***Debug*** або ***Release.*** Глобально ці налаштування задаються для всієї програми в вікні ***Compiler and debugger settings*** пункту меню ***Settings→Compiler and debugger ...***

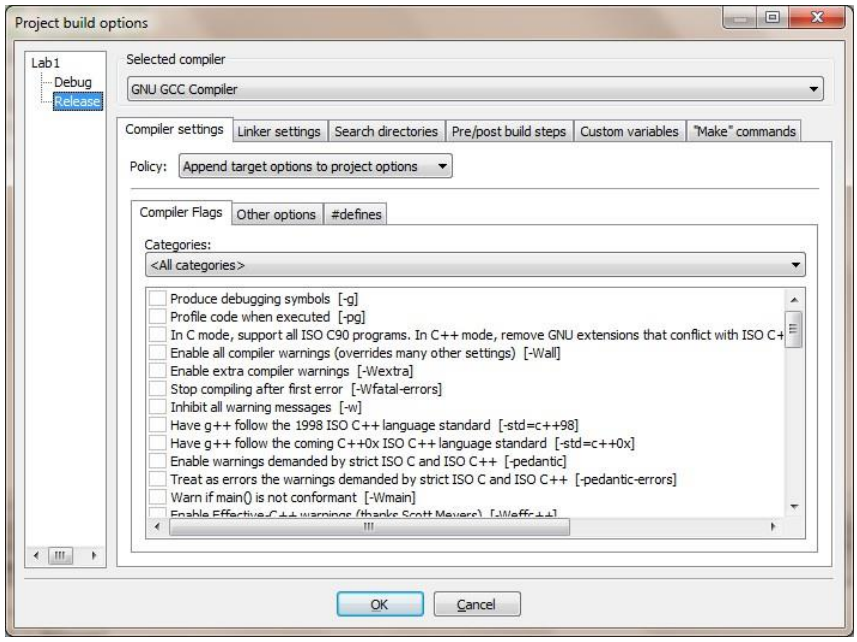

Рис. 1.21 Вікно настройки компіляції і зв'язування проекту ***Project build options***

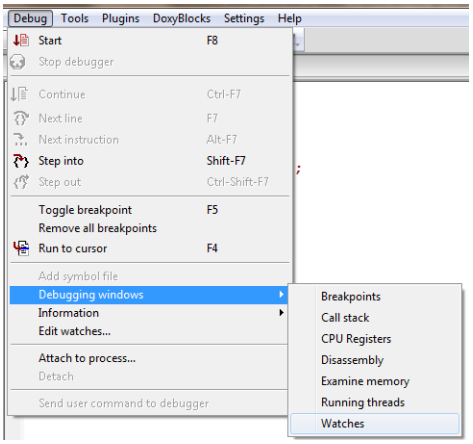

Рис. 1.22 Меню Debug

Відлагодження проекту (практичної роботи)

Для пошуку помилок виконання у програмі, тобто *багів* (історична назва з'явилася в 1940-их від англ. слова *bug*, або жук, комаха), використовується той відлагоджувач, що поставляється разом з компілятором. У даному випадку це Gdb. Для роботи з відлагоджувачем використовується меню ***Debug*** (рис. 1.22). Для запуску відлагоджувача потрібно вибрати пункт меню ***Debug→Start***. Після чого він запуститься, і якщо не вказано точок зупинки, то відлагоджувач відпрацює пройшовши по кроках всю програму та у випадку, коли не трапиться аварійна зупинка – завершиться. Для перегляду вмісту змінних, треба вказати точки зупинки, які відзначаються червоними крапками (рис. 1.23).

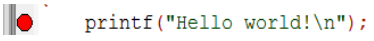

Рис. 1.1 Приклад точки зупинки

Точка зупинки встановлюється або натисканням лівої кнопки миші по сірій розділовій смузі поряд з номером рядка, або установкою курсору на рядок, в якому треба зупинитися. Відлагоджувач працює тільки у ***Debug*** збірці програми! Для перегляду вмісту змінних і масивів використовується панель (вікно) спостереження змінних ***Watches*** (рис. 1.24).

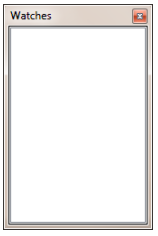

Рис. 1.24 Панель спостереження ***Watches***

Щоб явно додати в це вікно зміні, за якими необхідно спостерігати, треба натиснути по вікну ***Watches*** правою кнопкою миші, після чого з'явиться меню, що випадає, де треба вибрати пункт ***Add watch***. Після цього відкриється діалогове вікно ***Edit watch*** (рис. 1.25), в якому в полі ***Keyword*** треба ввести ім'я змінної. Після чого натиснути кнопку ***OK.*** Якщо потрібно переглянути масив, то слід відзначити прапорцем ***Watch as array.***

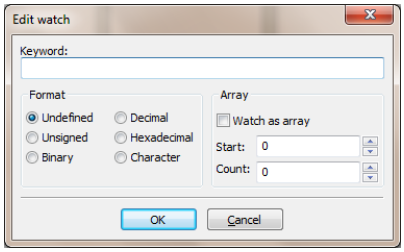

Рис. 1.25 Діалогове вікно ***Edit watch***

Вікно ***Watches*** автоматично відображатиме локальні змінні та аргументи функцій (рис. 1.26).

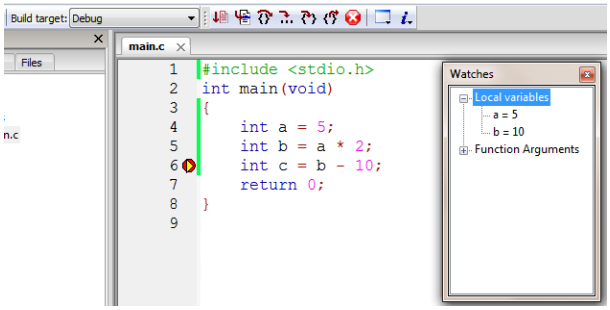

Рис. 1.26 Приклад відлагодження програми

1.2.4.Hello World – перша консольна програма

Консольна програма орієнтована на символьний ввід-вивід, що робить її корисною при вивченні стандартних функцій вводу-виводу мови С.

Програма *Hello world* – традиційна серед програмістів перша програма, що наводиться в підручниках з мов програмування. Програма виводить рядок "Hello, world!" або його еквіваленти.

Незважаючи на свою простоту, програма корисна тим, що дає змогу початківцю виконати всі дії, необхідні для написання, компілювання і запуску простої програми, написаної обраною мовою програмування.

На рис. 1.27 зображено код програми "Hello world!" на мові С[1].

---

[1] Увімкнути відображення номерів рядків можна у вікні ***General settings*** (***Settings→Editor…***) у вкладці ***Editor settings… / Other options / Show line numbers***.

```c
#include <stdio.h>
#include <conio.h>
int main(void)
{
    printf("Hello world!\n");
    getch();
    return 0;
}

```

Рис. 1.27 Код програми "Hello world!" на мові С

Перші два рядки це директиви препроцесора, що вказують необхідність підключення стандартних бібліотек. Бібліотека **<stdio.h>** тобто бібліотека стандартного вводу-виводу містить функцію **printf()**, що використовується для виводу текстових повідомлень на консоль. Бібліотека **<conio.h>**, тобто бібліотека консольного вводу-виводу містить функцію **getch()**, що відповідає за зчитування символу з клавіатури. Третій рядок оголошує головну функцію програми, яка завжди має назву **main**. З цієї функції починається будь-яка програма на мові С. Сьомий рядок містить оператор повернення **return**, який повертає число нуль після успішного завершення функції **main**, тобто після успішного завершення програми[1].

Для компілювання програми слід виконати команду ***Build***. Після цього компілятор створить у каталозі проекту об'єктний файл під назвою, що відповідає назві файлу з текстом програми, а компонувальник збере цей файл та інші необхідні об'єктні файли у виконуваний модуль під назвою, що відповідає назві проекту (рис. 1.28).

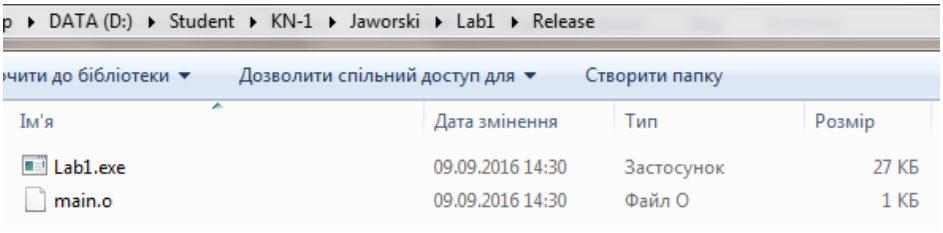

Рис. 1.28 Результати компілювання та компонування програми

На рис. 1.29 зображено результати виконання програми. В консольне вікно виводиться відповідне текстове повідомлення, після чого очікується введення будь-якого символу з клавіатури.

---

[1] Історично склалося так, що в разі успішного виконання програма та досить широке коло стандартних функцій повертають у місце свого виклику число нуль. У стандартній бібліотеці **<stdlib.h>** для цього спеціально відведена відповідна директива **EXIT_SUCCESS**. Така функціональна можливість використовувалася для аналізу ходу виконання програм до появи новіших інструментів, таких як наприклад обробники виняткових ситуацій.

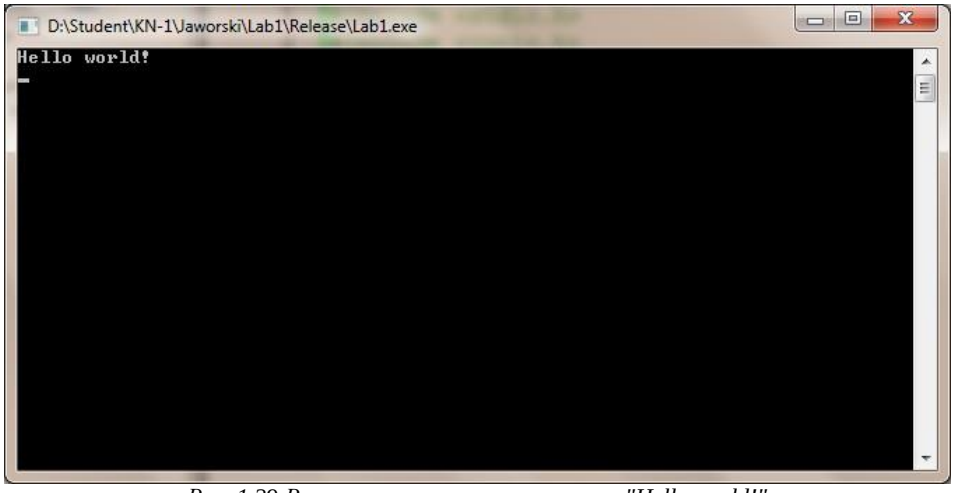

Рис. 1.29 Результати виконання програми "Hello world!"

1.2.5.Використання компілятора та компонувальника без IDE

На практиці трапляються випадки, коли необхідно скомпілювати та скомпонувати коди програми на комп'ютерах, де відсутні звичні інтегровані середовища розробки програмних продуктів. Наприклад в Linux системах компілятор мови С поставляється за замовчуванням, а для редагування тексту програм можна використати стандартний текстовий редактор.

Компілятор та компонувальник це також програми, що виконуються аналогічно до інших в конкретній операційній системі (інколи це одна і та ж програма, що виконує різні дії в залежності від вхідних параметрів). У Code::Blocks шляхи до виконуваних файлів компілятора, компонувальника та відлагоджувача можна редагувати глобально в вікні ***Compiler and debugger settings*** пункту меню ***Settings → Compiler and debugger ...*** у вкладці ***Toolchain executables*** (рис. 1.30)[1].

Виконуючи команду ***Build*** для проекту інтегроване середовище розробки програмного забезпечення засобами операційної системи по черзі запускає програми з вхідними параметрами, що відповідають назвам файлів проекту. Ці дії можна зробити вручну з консолі виконавши команди[2]:

```
set path=E:\CodeBlocks-EP\MinGW\bin\ cd
D:\student\kn-1\jaworski\lab1 mingw32-
gcc.exe main.c -o lab1.exe lab1.exe
```

[1] У деяких випадках, для можливості використання функціональних можливостей нових стандартів мови С99 та новіших років необхідно буде явно вказати параметр компілятора "**`-std=c99`**" у вікні ***Compiler and debugger settings*** вкладці ***Compiler settings / Other options***.

[2] Наведено приклад консольних команд лінійки операційних систем Windows.

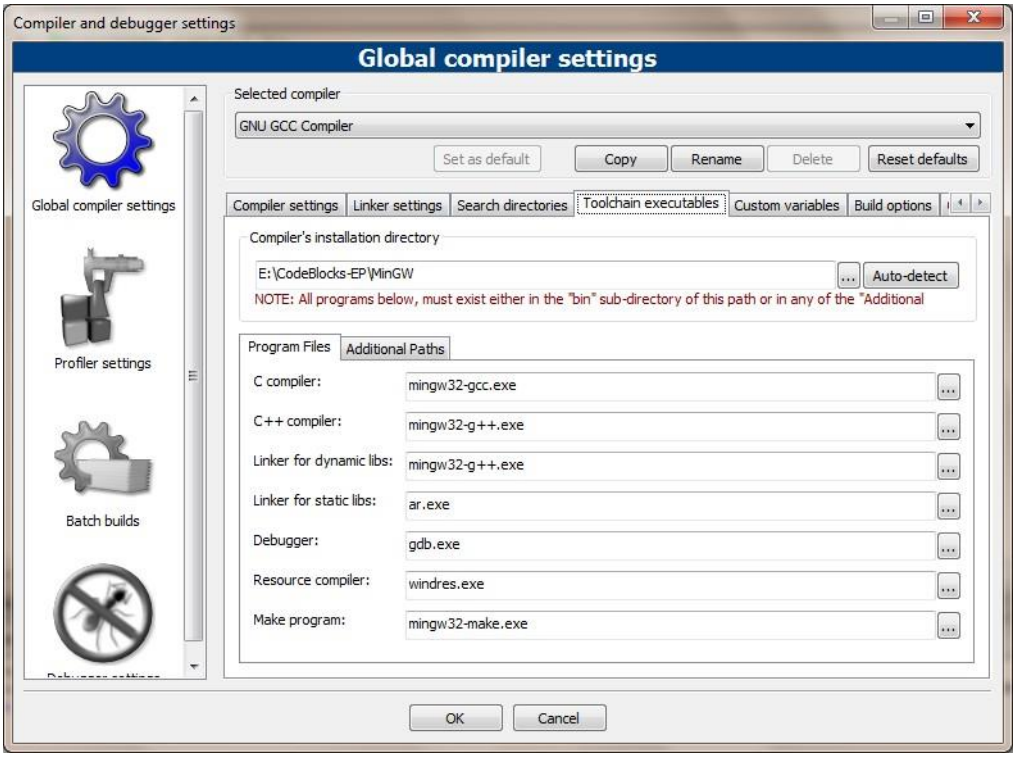

Рис. 1.30 Вкладка ***Toolchain executables***

У результаті отримаємо зібрану програму (рис. 1.31).

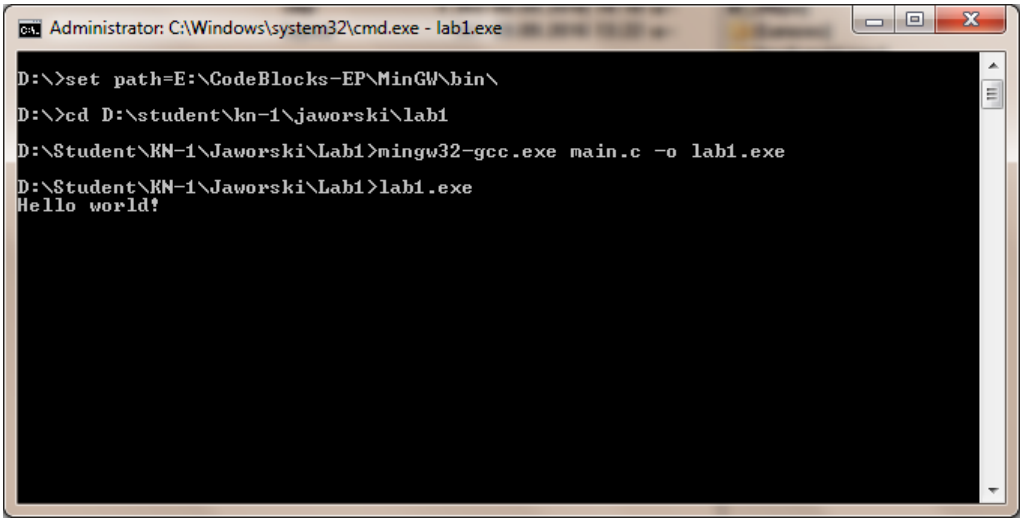

Рис. 1.31 Компіляція та компонування програми "Hello world!"
з допомогою консольних команд операційної системи Windows та компілятора MinGW

1.3. Контрольні запитання

1. Які є сім основних кроків для розроблення програм?
2. Що робить компілятор?
3. Що робить компонувальник?
4. Що робить відлагоджувач?
5. Чим об’єктний код відрізняється від виконуваного?
6. Що таке процедурне програмування?
7. Що таке інтегроване середовище розробки програмного забезпечення?
8. Які дії необхідно виконати для створення консольного проекту в інтегрованому середовищі розробки Code::Blocks?

1.4. Практичне завдання

1. Встановити середовище Code::Blocks (або можете скористатись іншим IDE на ваш розсуд) на Ваш ПК.
2. Запустити середовище Code::Blocks (або іншу IDE на Ваш розсуд), розглянути всі команди, запам’ятати найбільш необхідні для роботи команди.
3. Створити каталог з назвою свого прізвища на робочому диску. В ньому створити консольний проект під назвою "**`Lab1`**". Усі назви не повинні містити в собі пробіли, крапки, коми та інші розділові знаки.
4. У проекті створити файл коду програми на мові С.
5. Написати код програми "Hello world!", скомпілювати, скомпонувати та запустити програму до виконання.
6. Модифікувати програму так, щоб вона виводила на консоль ваше прізвище.
7. Скомпілювати, скомпонувати та запустити на виконання програму з допомогою консольних команд.

1.5. Зміст звіту

1. Титульний аркуш.
2. Мета роботи.
3. Хід виконання практичного завдання.
4. Текст модифікованої програми.
5. Результати виконання модифікованої програми.
6. Аналіз результатів та висновки.