

Підручник з KDevelop

Цю документацію було створено на основі сторінки
UserBase KDE KDevelop4/Manual.
Переклад українською: Юрій Чорноіван



Підручник з KDevelop

Зміст

1	Для чого призначено KDevelop?	6
2	Сеанси і проекти: основи KDevelop	8
2.1	Термінологія	8
2.2	Налаштування сеансу та імпортування вже створеного проекту	8
2.2.1	1. Імпортування даних проекту з сервера системи керування версіями	9
2.2.2	2. Імпортування проекту, дані якого вже зберігаються на жорсткому диску вашого комп'ютера	10
2.3	Налаштування програми як другого проекту	10
2.4	Створення проектів «з нуля»	10
3	Робота з кодом програм	12
3.1	Інструменти та панелі перегляду	12
3.2	Огляд роботи з кодом	14
3.2.1	Локальні дані	14
3.2.2	Дані щодо окремих файлів	16
3.2.3	Дані щодо проектів та сеансів	17
3.2.4	Пояснення щодо кольорів підсвічування	19
3.3	Пересування між фрагментами коду	19
3.3.1	Пересування поточним фрагментом	19
3.3.2	Пересування файлами та режим огляду	20
3.3.3	Пересування проектами та сеансами: семантичне пересування	21
3.4	Введення коду	25
3.4.1	Автозавершення	25
3.4.2	Додавання нових класів та реалізація вкладених функцій	27
3.4.3	Оголошення для документування	31
3.4.4	Перейменування змінних, функцій і класів	34
3.4.5	Фрагменти коду	35
3.5	Режими і робочі набори	37
3.6	Деякі корисні клавіатурні скорочення	39
4	Створення коду за допомогою шаблонів	41
4.1	Створення класу	41
4.2	Створення тесту модуля	43
4.3	Інші файли	43
4.4	Керування шаблонами	44

5	Збирання (компіляція) проєктів з нетиповими Makefile	46
5.1	Збирання окремих цілей з Makefile	46
5.2	Вибір збірки цілей з Makefile для регулярного збирання	47
5.3	Обробка повідомлень про помилки	48
6	Запуск програм у KDevelop	49
6.1	Налаштування запуску у KDevelop	49
6.2	Деякі корисні клавіатурні скорочення	50
7	Налагоджування програм у KDevelop	52
7.1	Запуск програми під керуванням програми для налагоджування	52
7.2	Приєднання програми для налагоджування до запущеного процесу	54
7.3	Деякі корисні клавіатурні скорочення	54
8	Робота з системами керування версіями	56
9	Налаштування KDevelop	58
9.1	Налаштування редактора	58
9.2	Налаштування відступів у коді	58
9.3	Налаштування клавіатурних скорочень	60
9.4	Налаштування автодоповнення коду	60
10	Подяки і ліцензія	62

Анотація

KDevelop — комплексне середовище розробки, призначене для виконання широкого кола завдань з програмування.

Розділ 1

Для чого призначено KDevelop?

KDevelop — сучасне комплексне середовище для розробки (IDE) мовою C++ та іншими мовами. Це середовище є частиною проєкту з розробки **стільничного середовища KDE**. Через це з середовищем можна працювати у Linux[®] (навіть у інших робочих середовищах, наприклад, GNOME), а також у інших варіантах UNIX[®] та у Windows.

У KDevelop передбачено всі можливості сучасних комплексних середовищ розробки. Для роботи з великими проєктами та програмами найважливішою можливістю є те, що KDevelop *розуміє C++*: середовище виконує обробку всієї кодової бази і запам'ятовує елементом яких класів є кожна з функцій, де визначено і яким є тип кожної зі змінних, а також багато інших параметрів вашого коду. Наприклад, нехай у одному з файлів заголовків вашого проєкту визначено клас

```
class Car {
    // ...
    public:
        std::string get_color () const;
};
```

а пізніше у програмі використано такий код:

```
Car my_ride;
// ...якісь дії з цією змінною...
std::string color = my_ride.ge
```

середовище запам'ятає, що фрагмент `my_ride` у останньому рядку є змінною типу `Car`, отже запропонує вам доповнення коду `ge` у форматі `get_color()`, оскільки ця функція є єдиною функцією класу `Car`, назва якої починається з «ge». Замість введення повної назви функції вам достатньо натиснути **Enter**, щоб отримати ціле слово. Таким чином, ви можете зекономити час і уникнути неприємних друкарських помилок, крім того, вам не потрібно буде запам'ятовувати точні назви тисяч функцій і класів, з яких складаються великі проєкти.

Для другого прикладу використаємо такий код:

```
double foo ()
{
    double var = my_func();
    return var * var;
}
double bar ()
{
    double var = my_func();
    return var * var * var;
}
```

Якщо ви наведете вказівник миші на символ `var` у функції `bar`, середовищем буде пункт для показу всіх використань цього символу. Якщо ви натиснете цей пункт, середовище покаже всі використання змінної у функції `bar`, оскільки KDevelop розуміє, що змінна `var` у функції `foo` це зовсім інша змінна. Крім того, клацання правою кнопкою миші на назві змінної надасть вам змогу перейменувати її. Середовище виконає заміну змінної лише у функції `bar`, але не чіпатиме зміну з тією самою назвою у функції `foo`.

KDevelop не лише редактор коду з елементами штучного інтелекту, KDevelop дуже добре виконує інші дії. Звичайно ж, середовище підсвічує код різними кольорами; передбачено інструмент керування відступами, вбудований інтерфейс зневадника GNU `gdb`; середовище здатне показувати документацію до функції, якщо ви наведете вказівник миші на запис функції у коді; середовище може працювати з різними середовищами збирання та компіляторами (наприклад, з проєктами, заснованими на **make** та **cmake**), а також вміє ще багато чого, що ми і обговоримо у цьому підручнику.

Розділ 2

Сеанси і проєкти: основи KDevelop

У цьому розділі ми оглянемо деякі питання щодо термінології, використаної у KDevelop, та структури роботи у середовищі. Зокрема, ми поговоримо про поняття *сеанси* і *проєкти* та пояснимо, як налаштувати проєкти, над якими ви маєте намір працювати у KDevelop.

2.1 Термінологія

У KDevelop використано поняття *сеанси* та *проєкти*. У сеансі містяться всі проєкти, чимось пов'язані між собою. У наведених нижче прикладах ми припускаємо, що ви є розробником одночасно бібліотеки та програми, яка використовує цю бібліотеку. Прикладом подібної схеми розробки є бібліотека KDE: (бібліотека) і саме середовище KDevelop (програма). Інший приклад: ви є розробником ядра Linux[®] і одночасно працюєте над драйвером пристрою для Linux[®], який ще не включено до ядра.

У останньому прикладі у нас був би сеанс KDevelop з двома проєктами: ядром Linux[®] і драйвером пристрою. Варто згрупувати ці проєкти у одному сеансі (замість двох сеансів для кожного окремого проєкту), оскільки корисно бачити функції та структури даних ядра у KDevelop, коли ви пишете код драйвера: ви, наприклад, зможете скористатися автодоповненням назв функцій та змінних ядра або переглянути документацію з функції ядра під час розробки драйвера пристрою.

Припустімо тепер, що ви є одним з розробників KDE. Тоді у вас буде інший сеанс, який міститиме проєкт KDE. Звичайно, ви можете створити єдиний сеанс для всіх ваших проєктів, але для цього немає ніяких причин: для вашої роботи у KDE вам не потрібен доступ до функцій ядра або драйверів пристроїв, вам не потрібне автоматичне доповнення назв класів KDE під час роботи над ядром Linux[®]. Нарешті, збирання бібліотек KDE не пов'язане зі збиранням ядра Linux[®] (з іншого боку збирання драйвера пристрою часто пов'язане зі збиранням ядра Linux[®], якщо було внесено зміни до файлів заголовків ядра).

Нарешті, ще одним з випадків використання сеансів є одночасна робота над експериментальною версією проєкту та його стабільною версією: у цьому випадку небажаним буде конфлікт між класами, які належать різним гілкам проєкту у KDevelop. Отже, варто створити два сеанси з однаковим набором проєктів, але різними каталогами зберігання (відповідно до гілок розробки).

2.2 Налаштування сеансу та імпортування вже створеного проєкту

Зупинимось на прикладі з ядром Linux[®] і драйвером пристрою. Вам слід буде замінити назви бібліотек та проєктів вашими назвами бібліотек і проєктів, щоб реалізувати наші приклади

у вашій системі. Щоб створити сеанс, який міститиме два наших проекти, скористаємося пунктом меню **Сеанс** → **Почати новий сеанс**, розташованим вгорі ліворуч (або якщо ви вперше запустили KDevelop, просто скористайтеся типовим сеансом, його буде відкрито по-рожнім).

Далі нам потрібно заповнити сеанс проектами, які у нашому прикладі будуть вже створеними раніше проектами (створення проектів «з нуля» обговорено у іншому розділі цього підручника). Виконати заповнення можна у два способи. Один з них можна застосувати до проектів, дані яких вже зберігаються на жорсткому диску вашого комп'ютера. Іншим можна скористатися для отримання даних проекту з сервера.

2.2.1 1. Імпортування даних проекту з сервера системи керування версіями

Припустімо, що дані потрібного нам проекту, — скажімо, ядра Linux[®], — зберігаються у якійсь системі керування версіями, але у вас ще немає копії сховища коду на жорсткому диску комп'ютера. У такому разі, відкрийте меню **Проект** для створення проекту ядра Linux[®] у межах поточного сеансу і виконайте такі дії:

- Скористайтеся пунктом меню **Проект** → **Отримати проект** для імпортування даних проекту.
- Середовище запропонує вам розпочати новий проект у межах поточного сеансу, залежно від походження коду: ви можете просто вказати KDevelop вже створений каталог з кодом, ви також можете наказати KDevelop отримати код зі сховища коду.
- Припускаємо, що у вас ще немає копії коду зі сховища системи керування версіями. Вам слід виконати такі дії:
 - У діалоговому вікні під написом **Вибір джерела** виберіть **З файлової системи, Subversion, Git, GitHub** або **KDE**.
 - Виберіть робочий каталог призначення, до якого слід отримати код проекту.
 - Виберіть адресу розташування сховища з кодом проекту.
 - Натисніть кнопку **Отримати**. Отримання кодів може бути доволі тривалою справою. Тривалість виконання цієї дії залежить від ширини каналу вашого з'єднання з інтернетом та розміру проекту. На жаль, у KDevelop 4.2.x панель поступу не є надто інформативною, але ви можете спостерігати за поступом з командного рядка за допомогою команди

```
du -sk /шлях/до/проекту/KDevelop
```

(буде показано об'єм отриманих даних).

ПРИМІТКА

Розробникам відомо про проблему зі смужкою поступу: [KDevelop, вада 256832](#).

ПРИМІТКА

Під час обробки ви можете побачити повідомлення щодо помилки: «Вам слід вказати коректне розташування проекту». Можете не зважати на це повідомлення.

- Середовище попросить вас вибрати файл проекту KDevelop у каталозі з кодом. Оскільки такого файла, ймовірно, у вас ще немає, просто натисніть кнопку **Далі**.
- Ще раз натисніть кнопку **Далі**.

- KDevelop попросить вас визначитися зі способом керування проектом. Якщо у проекті використано стандартні файли make UNIX®, виберіть нетиповий спосіб керування проектом за допомогою файла makefile.
- KDevelop виконає обробку даних проекту. Знову ж таки, ця дія може бути доволі тривалою, оскільки доведеться виконати обробку всіх файлів, створити покажчик класів тощо. У нижній правій частині основного вікна буде показано панель поступу з даними щодо поступу виконання обробки. (Якщо у вашого процесора декілька ядер, ви можете пришвидшити процедуру: скористайтеся пунктом меню **Параметри** → **Налаштувати KDevelop**, виберіть пункт **Інструмент фонові обробки** на панелі ліворуч і збільшіть кількість потоків обробки у полі праворуч.)

2.2.2 2. Імпортування проекту, дані якого вже зберігаються на жорсткому диску вашого комп'ютера

Якщо ж проект, над яким ви бажаєте працювати, вже зберігається на жорсткому диску вашого комп'ютера (наприклад, ви отримали дані проекту у архіві з сервера FTP, скопіювали зі сховища керування версіями проекту або ви працюєте над власним проектом, дані якого зберігаються *лише* на жорсткому диску вашого комп'ютера), скористайтеся пунктом меню **Проекти** → **Відкрити/Імпортувати проект** і у діалоговому вікні вкажіть каталог, у якому зберігаються дані вашого проекту.

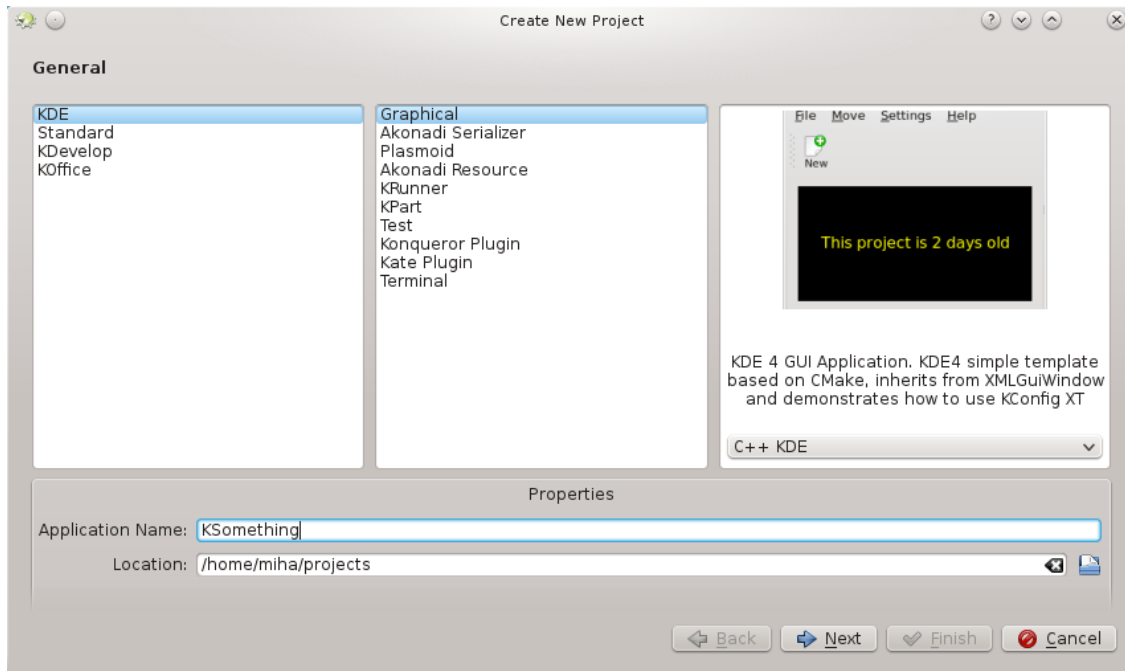
2.3 Налаштування програми як другого проекту

Наступним кроком буде, очевидно, налаштування інших проектів у тому самому сеансі. У нашому раніше наведеному прикладі другим проектом буде драйвер ядра. Додати новий проект можна знову виконавши вказані вище кроки.

Якщо ви розробляєте одночасно декілька програм або бібліотек, просто повторіть вказані вище кроки для додавання цих проектів до вашого сеансу.

2.4 Створення проектів «з нуля»

Звичайно ж, ви можете скористатися новим проектом. Створити новий проект можна за допомогою пункту меню **Проекти** → **Створити за шаблоном**, який відкриває діалогове вікно вибору шаблону. Деякі з шаблонів встановлюються з основним пакунком KDevelop, інші ж можна встановити разом з програмою KAppTemplate. Виберіть у діалоговому вікні тип проекту і мову програмування, вкажіть назву і розташування вашого проекту і натисніть кнопку **Далі**.



За допомогою другої сторінки діалогового вікна ви можете налаштувати систему керування версіями. Виберіть бажану для вас систему і заповніть відповідні поля параметрів. Якщо ви не хочете користуватися системою керування версіями або хочете визначити її параметри пізніше, виберіть пункт **Немає**. Щойно визначення відповідних параметрів буде завершено, можете натиснути кнопку **Завершити**.

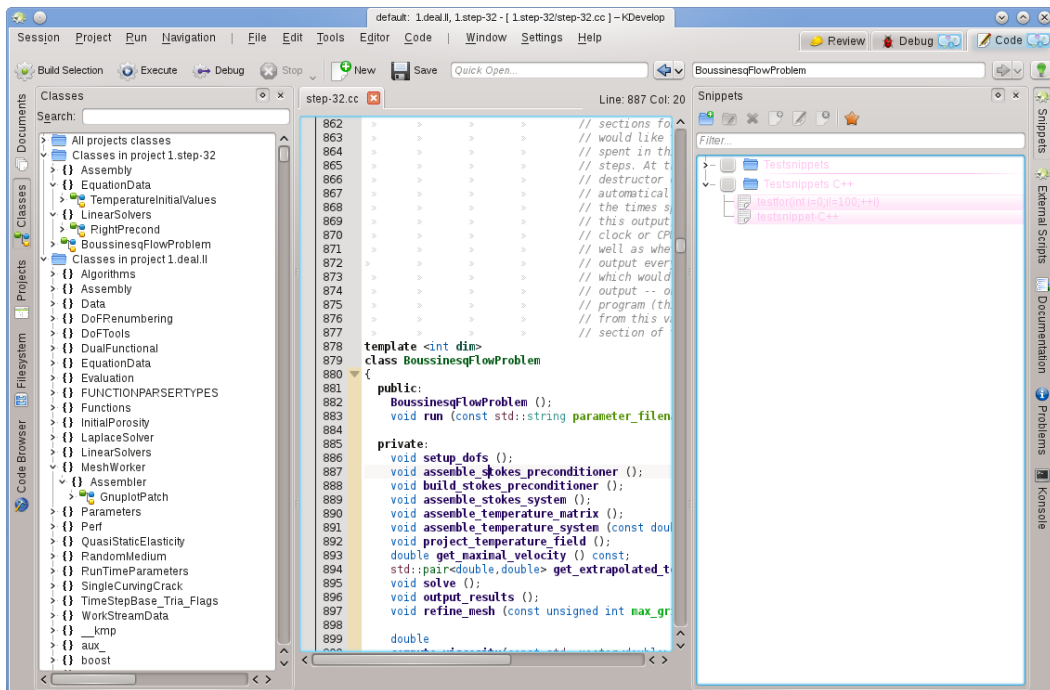
Тепер ваш проєкт створено, ви можете спробувати зібрати і встановити його. У деяких шаблонах передбачено коментарі у кодї або навіть окремий файл README. Рекомендуємо ознайомитися зі вмістом такого файлу до того, як буде розпочато подальшу розробку. Після ознайомлення з усіма довідковими матеріалами можна розпочати роботу над проєктом і додавання потрібних вам можливостей.

Розділ 3

Робота з кодом програм

Окрім налагоджування, читання та написання коду є найважливішими завданнями під час розробки програмного забезпечення. З метою полегшення навігації кодом та його написання у KDevelop передбачено багато різних інструментів. Як буде докладніше показано у наступних розділах, KDevelop не просто редактор коду, — це скоріше система керування кодом, яка може подавати різні дані, отримані на основі аналізу загальної сукупності коду всього вашого сеансу роботи.

3.1 Інструменти та панелі перегляду



Для роботи з проектами у KDevelop передбачено *інструменти*. Інструмент надає певні дані щодо коду або виконує з ним певну дію. Інструментам відповідають кнопки вздовж периметра вікна програми (з вертикальним текстом на полях ліворуч і праворуч та горизонтальним вздовж нижнього поля). Якщо ви натиснете таку кнопку, у головному вікні буде відкрито

підвікно — *панель перегляду*; якщо кнопку буде натиснуто ще раз, відповідне підвікно буде закрито.

Щоб закрити допоміжне вікно, також можна натиснути кнопку **x** у верхній правій частині цього вікна.

На наведеному вище зображенні ви можете бачити певний набір інструментів, вирівняних за лівим і правим полем. Ліворуч відкрито панель інструмента **Класи**, праворуч — **Фрагменти**. Посередині можна бачити панель редактора. З практичних міркувань, переважну частину часу розробки варто працювати лише з редактором та панеллю **Класи** чи **Перегляд коду**, відкритою ліворуч. На час використання певного інструмента можна відкрити його панель, яку після використання варто закрити з метою збільшення простору для панелі редактора.

Після першого запуску KDevelop ви вже зможете скористатися кнопкою інструмента **Проекти**. Натисніть цю кнопку: у відповідь буде відкрито панель зі списком проектів, які було додано до сеансу у нижній частині вікна та панель перегляду файлової системи вашого проекту у верхній його частині.

У KDevelop ви можете скористатися багатьма іншими інструментами, не всі з них представлені кнопками вздовж периметра вікна у початковому стані. Щоб додати кнопку, скористайтеся пунктом меню **Вікна** → **Додати панель інструмента**. Ось перелік можливих корисних інструментів:

- **Класи**: повний список всіх класів, які визначено у одному з проектів або у вашому сеансі з усіма вбудованими функціями та змінними. Натискання пункту елемента класу відкриє вікно редактора з місцем оголошення елемента, пункт якого було натиснуто.
- **Документи**: містить список нещодавно відкритих файлів за типами (наприклад файли з кодом, файли латок, звичайні текстові документи).
- **Перегляд коду**: залежно від розташування курсора у редакторі файла, на цій панелі буде показано пов'язані з кодом дані. Наприклад, якщо курсор перебуває у рядку з `#include`, на панелі буде показано дані щодо файла, який включено до коду, зокрема дані щодо оголошених у файлі класів. Якщо курсор перебуває на порожньому рядку у файлі, на панелі буде показано класи і функції, оголошені і визначені у поточному файлі (всі як посилання: натискання відповідного пункту відкриватиме файл з оголошенням або визначенням класу чи функції). Якщо курсор перебуває на визначенні функції, на панелі буде показано місце оголошення та список місць, у яких використано функцію.
- **Файлова система**: показує ієрархічну структуру файлової системи.
- **Документація**: надає вам змогу виконувати пошук даних на сторінках довідника (man) та у інших довідкових документах.
- **Фрагменти**: на цій панелі буде наведено фрагменти тексту, які використовують вами регулярно, і які ви не хочете повторно вводити кожного разу. Наприклад, на основі якого було створено знімок екрана, була потреба часто використовувати фрагмент коду

```
for (typename Triangulation< dim>::active_cell_iterator cell
    = triangulation.begin_active();
    cell != triangulation.end();
    ++cell)
```

Вираз доволі незграбний, але використовується майже у такій формі кожного разу, коли вам потрібен цикл, — чудовий кандидат на включення до списку фрагментів.

- **Konsole**: відкриває панель командного рядка у головному вікні KDevelop, щоб ви могли віддати потрібну вам команду оболонки (наприклад, виконати `./configure`).

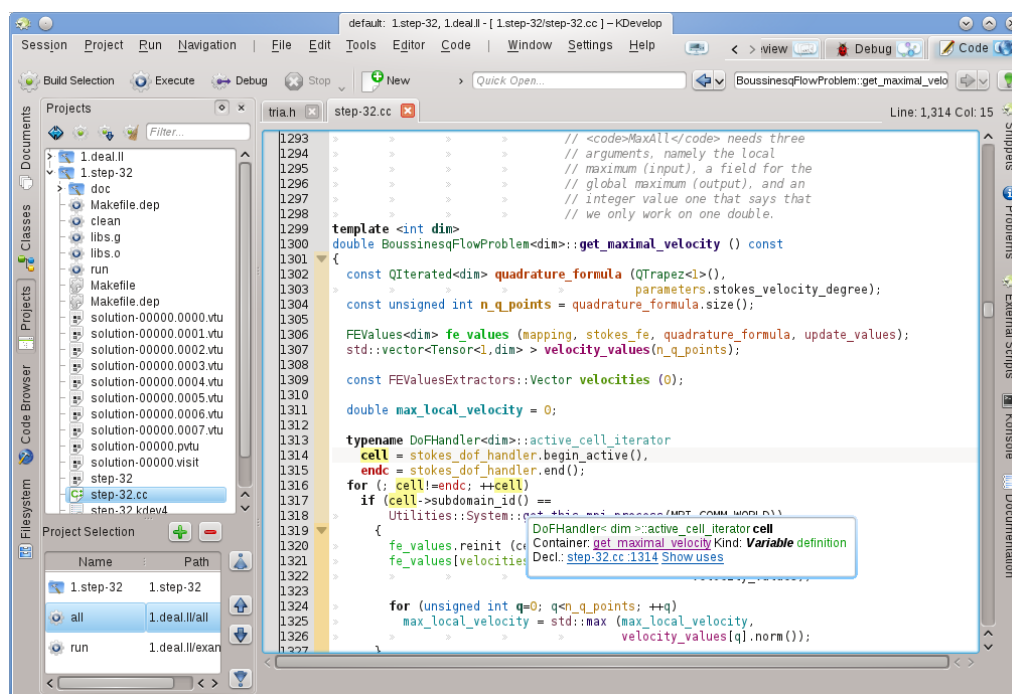
З повним списком інструментів та панелей перегляду можна ознайомитися [тут](#).

Для багатьох програмістів найважливішою є економія вертикального місця на екрані. Щоб досягти такої економії, ви можете розташувати панелі інструментів вздовж лівої та правої межі вікна програми. Щоб пересунути панель, клацніть на заголовку правою кнопкою миші і виберіть нове її розташування.

3.2 Огляд роботи з кодом

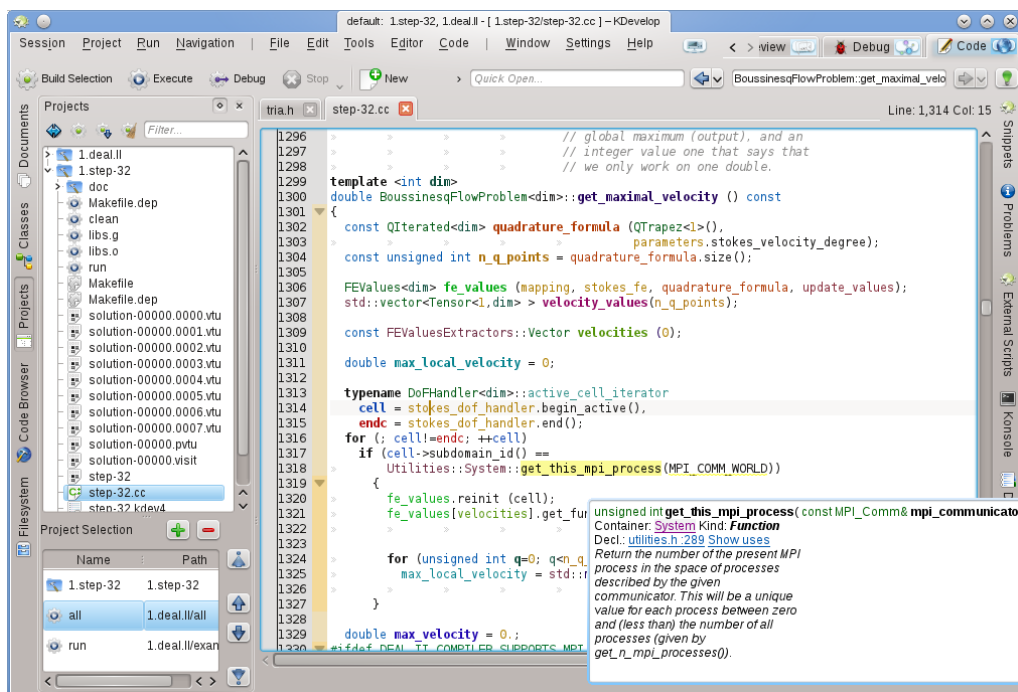
3.2.1 Локальні дані

KDevelop *розуміє* код програми, тому це середовище може надавати вам дані щодо змінних або функцій вашої програми. Наприклад, у цьому підручнику наведено знімок роботи з фрагментом коду, де вказівник миші наведено на символ `cell` у рядку 1316 (якщо ви надаєте перевагу роботі за допомогою клавіатури, того самого ефекту можна досягти утримуванням певний час натиснутою клавішею **Alt**):



KDevelop показує підказку, зокрема тип змінної (тут: `DoFHandler<dim>::active_cell_iterator`), де цю змінну оголошено (*контейнер*, яким тут є функція-обгортка `get_maximal_velocity`, оскільки це локальна змінна), тип даних (змінна, не функція, клас або простір назв) та місце оголошення (у рядку 1314, декілька рядків коду).

У поточному контексті з символом, на який наведено вказівник миші, не пов'язано жодної документації. У нашому прикладі, де вказівник миші наведено на символ `get_this_mpi_process` у рядку 1318, буде показано такі дані:

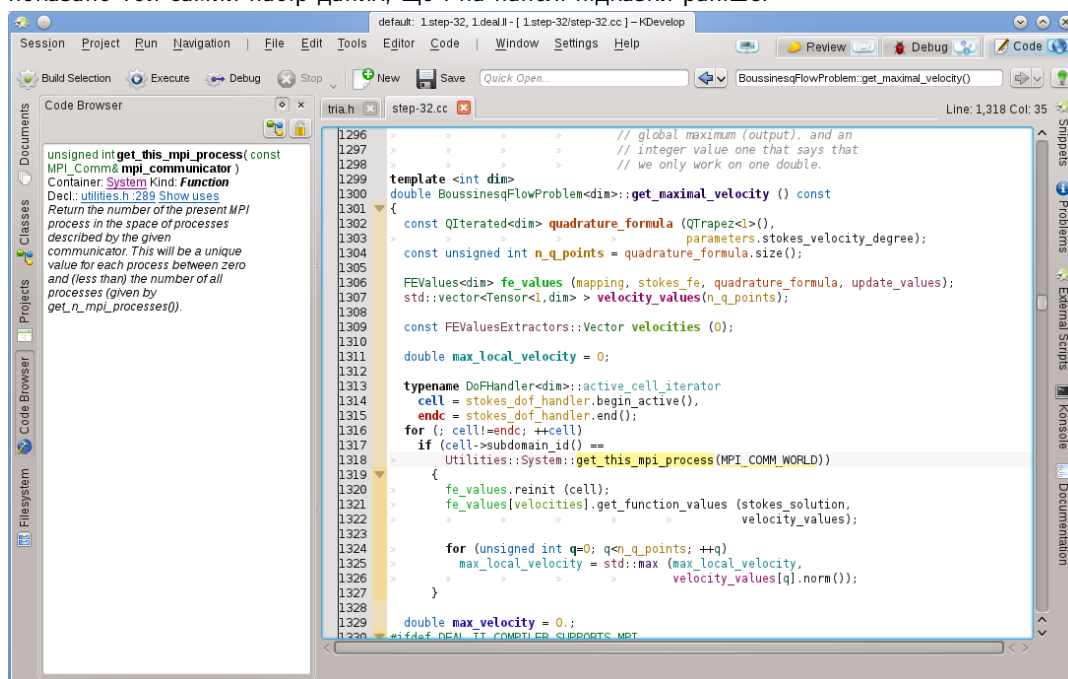


Тут KDevelop показано оголошення зі стороннього файлу (`utilities.h`, який є частиною зовсім іншого проекту того самого сеансу) разом з коментарем у форматі дохуген, який супроводжує цей оголошення.

Ще кориснішими підказки робить те, що вони є динамічними: можна натиснути пункт контейнера, щоб отримати дані щодо контексту, у якому оголошено змінну (тобто дані щодо простору назв `System`, зокрема місця його оголошення, визначення, використання та документації), і можна натиснути сині посилання, які повернуть курсор на позицію оголошення символу (наприклад, у `utilities.h`, рядок 289) або покажуть список місць, у яких використано символ у поточному файлі або усіх проектах поточного сеансу. Остання можливість буде корисною, якщо вам потрібно визначити, як, наприклад, певну функцію використано у об'ємному коді.

ПРИМІТКА

Панель підказки з часом зникатиме: щоб її знову відкрити доведеться знов утримувати натиснутою клавішу **Alt** або наводити вказівник миші на фрагменти коду. Якщо вам потрібно зафіксувати дані з цієї панелі, відкрийте панель інструмента **Перегляд коду**. У нашому прикладі курсор перебуває у тій самій функції, що і раніше, а на панелі інструмента ліворуч показано той самий набір даних, що і на панелі підказки раніше:



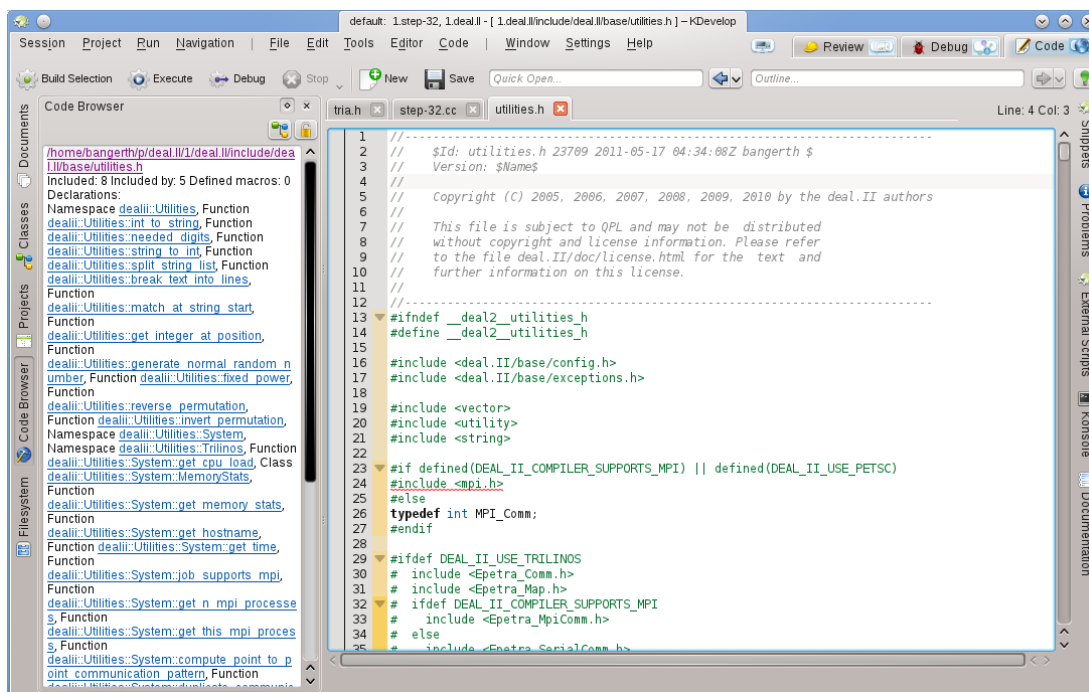
Пересування курсора у правій частині вікна призводитиме до зміни даних у його лівій частині. Натискання кнопки **Заблокувати поточний перегляд** у верхній правій частині надасть вам змогу зафіксувати дані, убезпечивши їх від зміни розташування курсора на час перегляду.

ПРИМІТКА

Доступ до контекстних даних такого типу можна отримати з багатьох частин KDevelop, не лише з панелі редактора коду. Наприклад, утримання натиснутою клавіші **Alt** у списку автоматичного доповнення (наприклад під час пришвидшеного відкриття якогось файла) також призводитиме до показу контекстних даних щодо поточного рядка.

3.2.2 Дані щодо окремих файлів

Наступним рівнем є отримання даних щодо всього файла коду, над яким ви працюєте. Щоб переглянути її, розташуйте курсор на початку поточного файла і подивіться на дані, які буде показано на панелі інструмента **Перегляд коду**:



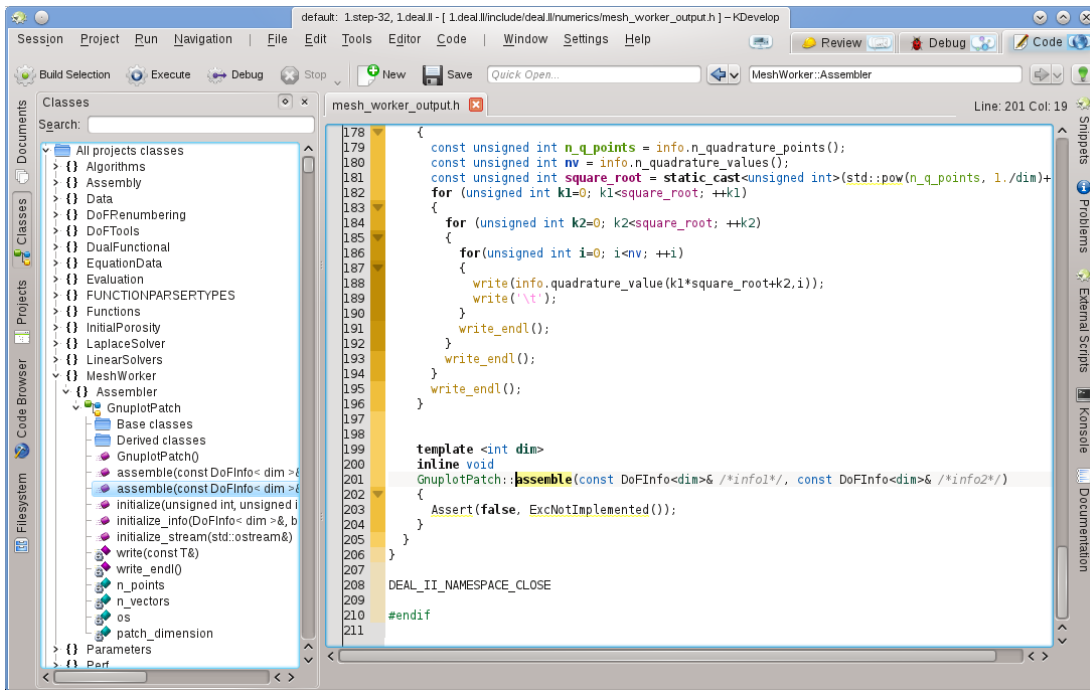
Тут середовищем показано список просторів назв, класів та функцій, оголошених або визначених у поточному файлі. За допомогою цього списку ви можете ознайомитися з загальними даними щодо дій, які виконуються у файлі, та безпосередньо перейти до будь-якого з оголошень або визначень без потреби у гортанні коду файла або пошуку певного фрагмента.

ПРИМІТКА
 Дані, які показано для всього файла, є тими самими, які буде показано у режимі «Огляд», обговорення якого у контексті навігації кодом викладено далі. Відмінність полягає у тому, що у режимі огляду ці дані буде показано лише на тимчасовій панелі підказки.

3.2.3 Дані щодо проєктів та сеансів

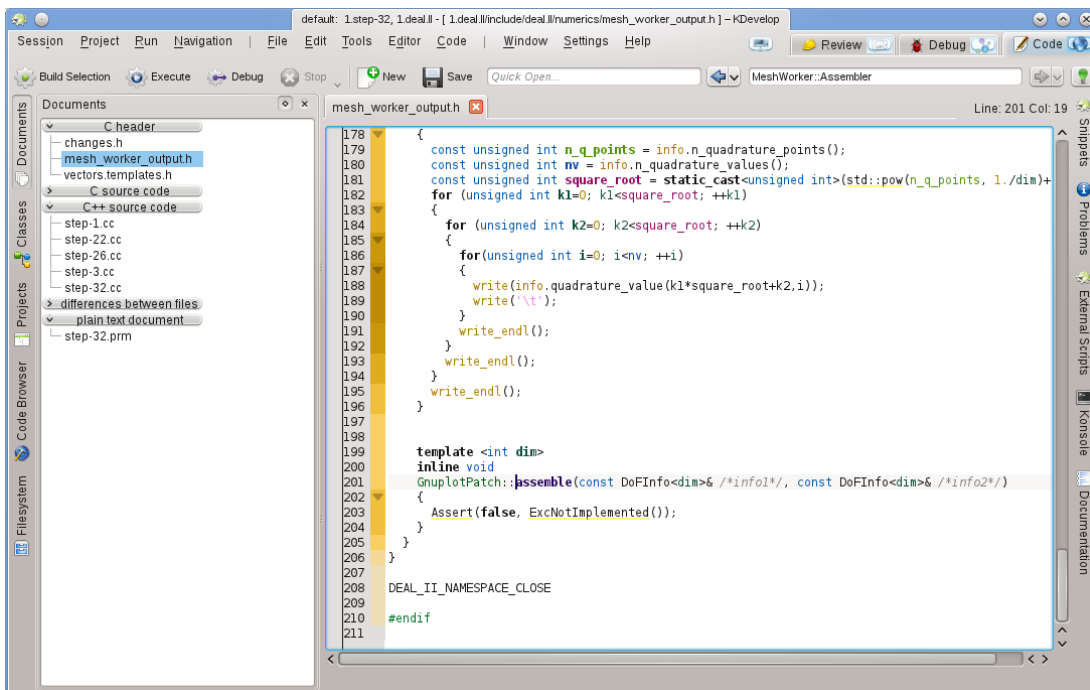
Існує багато способів отримати загальні дані щодо проєкту (або, фактично, всіх проєктів сеансу). Такі дані, зазвичай, можна знайти на панелях перегляду різноманітних інструментів середовища. Наприклад, на панелі інструмента **Класи** показано ієрархічну структуру всіх класів і зовнішніх просторів назв для всіх проєктів сеансу, разом з вбудованими функціями та змінними кожного з цих класів:

Підручник з KDevelop



Знову ж таки, за допомогою наведення вказівника миші на запис у списку можна отримати дані щодо відповідного символу, розташування його оголошення та визначення і випадків використання. Подвійним клацанням на пункті цього ієрархічного списку можна відкрити вікно редактора на позиції, де оголошено або визначено символ.

Але ви можете скористатися і іншими способами перегляду загальних даних. Наприклад, за допомогою інструмента **Документи** можна поглянути на проект з точки зору типів файлів або інших документів, з яких складається проект:



3.2.4 Пояснення щодо кольорів підсвічування

У KDevelop для підсвічування об'єктів у коді використовується ціла палітра кольорів. Якщо вам відоме значення цих кольорів, ви можете дуже швидко отримати доволі багато інформації щодо коду, просто подивившись на кольори, навіть не розбираючи кожен із символів коду окремо. Використано такі правила підсвічування:

- Об'єкти типів Class / Struct, Enum (значенні і тип), (загальні) функції та учасники класів буде позначено власним кольором (класи — зеленим, переліки (enum) — темно-червоним, а учасники — темно-жовтим або фіолетовим, (загальні) функції — завжди фіолетовим).
- Всі загальні змінні буде позначено темно-зеленим кольором.
- Ідентифікатори, які є визначеннями інших типів, буде позначено синьо-зеленим кольором.
- Всі оголошення та визначення об'єктів буде позначено напівжирним шрифтом.
- Якщо доступ до учасника класу здійснюється з контексту його визначення (базового або похідного класу), запису буде позначено жовтим кольором. Якщо ж доступ здійснюється поза цим контекстом, запису буде позначено фіолетовим кольором.
- Якщо учасник позначено як закритий (private) або захищений (protected), колір запису буде трохи темнішим, ніж зазвичай.
- Для локальних для вмісту функції змінних кольори вибиратимуться на основі хеш-коду ідентифікатора. Це ж стосується і параметрів функції. Ідентифікатор завжди матиме той самий колір у межах простору своєї дії, але той самий ідентифікатор матиме інший колір, якщо позначатиме інший об'єкт, тобто якщо його буде перевизначено на вкладеному рівні, отже ви зазвичай бачитимете для назви одного ідентифікатора у різних просторах однаковий колір. Отже, якщо у коді є декілька функцій, що приймають параметри з однаковими назвами, параметри цих функцій матимуть однаковий колір. Таке розфарбовування може бути окремо вимкнено за допомогою сторінки кольорів у діалоговому вікні параметрів програми.
- Ідентифікатори, для яких KDevelop не може визначити відповідне оголошення, буде позначено білим кольором. Таке позначення часто буває спричинено пропущеними інструкціями `#include`.
- Крім описано вище розфарбування, буде застосовано звичайні правила підсвічування синтаксичних конструкцій, визначені у Kate. Якщо виникатимуть конфлікти з підсвічуванням текстового редактора, завжди використовуватиметься семантичне підсвічування KDevelop.

3.3 Пересування між фрагментами коду

У попередньому розділі ми обговорювали вивчення коду програми, тобто отримання даних щодо символів, файлів та проєктів. Наступним кроком є перехід між компонентами коду, тобто навігація кодом. Знову ж таки, існує декілька рівнів, на яких можна здійснювати подібну навігацію: локально, у межах файла і у межах проєкту.

ПРИМІТКА

Доступ до багатьох засобів навігації можна отримати за допомогою меню **Навігація** головного вікна KDevelop.

3.3.1 Пересування поточним фрагментом

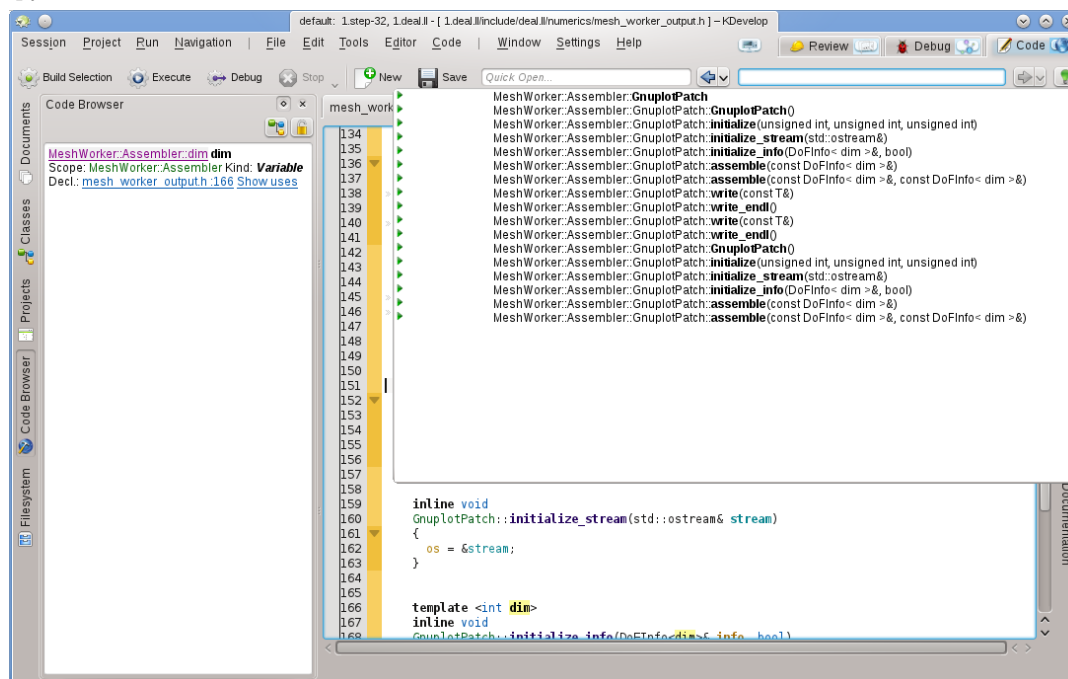
KDevelop — набагато більше, ніж просто редактор, — це *також* редактор коду. Тому, звичайно ж, ви можете пересувати курсор текстом за допомогою звичайних клавіш зі стрілочками. Ви також можете скористатися натисканням клавіш **PageUp** та **PageDown** і всіма іншими командами, якими можна скористатися у звичайному текстовому редакторі.

3.3.2 Пересування файлами та режим огляду

На рівні окремого файла у KDevelop передбачено багато способів навігації кодом програми. Приклад:

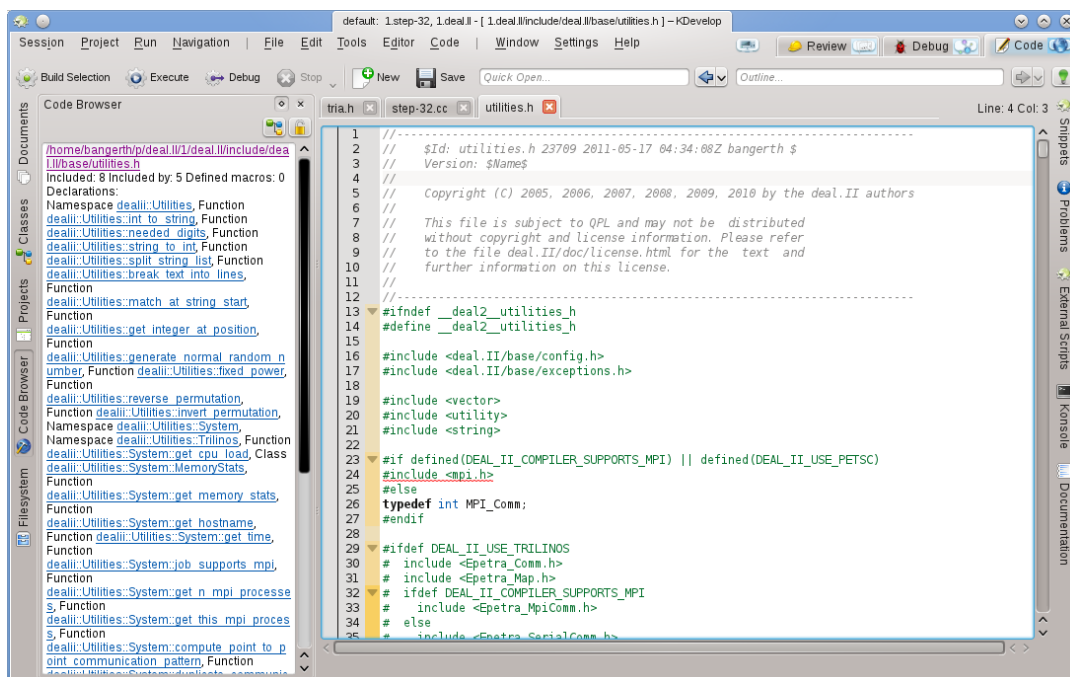
- **Огляд:** відкрити панель з оглядом вмісту поточного файла можна у принаймні три різних способи:

- Натисканням кнопки **Огляд** у верхній правій частині головного вікна або натисканням комбінації клавіш **Alt-Ctrl-N**. У відповідь буде відкрито спадне меню зі списком всіх функцій та оголошень класів:



Після цього ви можете вибрати пункт, до якого слід перейти, або — якщо таких пунктів багато, — почніть вводити текст, який може бути частиною потрібного пункту. У такому разі, введення символів у текстове поле скорочуватиме список, оскільки з нього виключатимуться пункти, які не відповідають введеним вами даним, доки список не буде скорочено до прийнятних розмірів.

- Розташуванням курсора у області файла (тобто поза межами оголошень та визначень функцій або класів) з відкритою панеллю інструмента **Перегляд коду**:



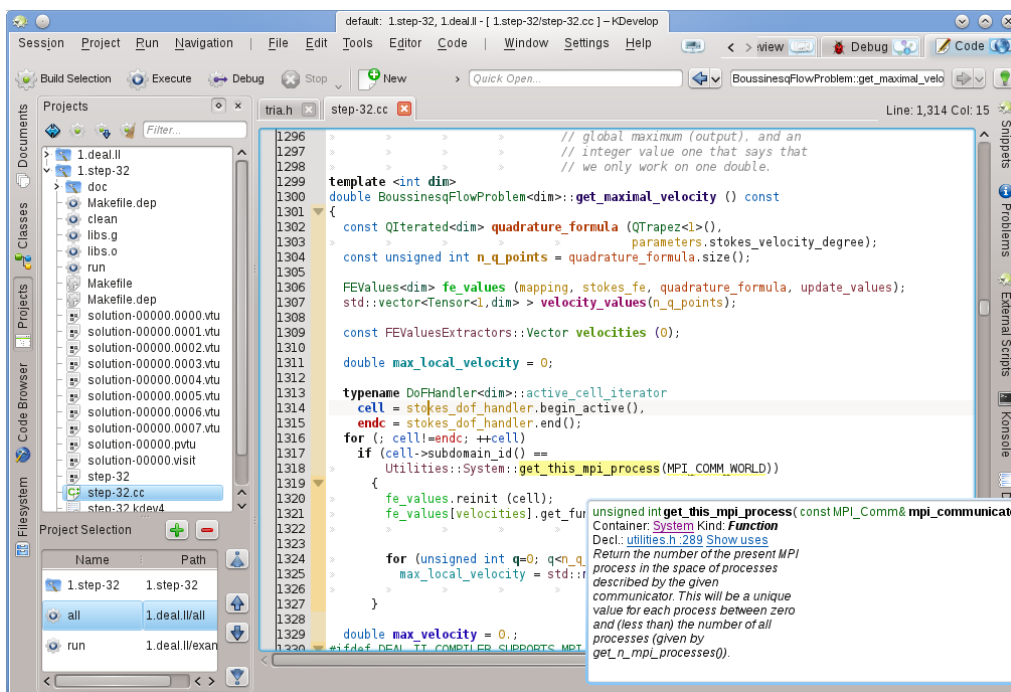
За показаною панеллю ви зможете стежити з даними поточного файла і надасть вам змогу вибрати пункт, до якого ви хочете перейти.

- Наведенням вказівника миші на вкладку одного з відкритих файлів. У відповідь буде показано панель з оглядом даних файла у відповідній вкладці.
- * Файли коду впорядковано у список оголошень та визначень. За допомогою натискання комбінацій клавіш **Alt-Ctrl-PgUp** і **Alt-Ctrl-PgDown** можна переходити до визначення попередньої або наступної функції у файлі.

3.3.3 Пересування проєктами та сеансами: семантичне пересування

Як ми вже згадували раніше, зазвичай KDevelop не працює з окремими файлами коду, замість цього середовище працює з цілими проєктами (або, точніше, з усіма проєктами, які є частиною поточного сеансу). Внаслідок цього середовище надає можливість навігації у межах цілих проєктів. Частина цих можливостей є наслідком можливостей, які ми вже обговорювали у розділі щодо [навігації кодом](#), інші ж є абсолютно відмінними від них. Основною особливістю цих можливостей навігації є те, що їх засновано на *семантичному розумінні* коду, тобто для роботи з ними потрібна певна обробка цілих проєктів та пов'язаних з ними даних. У наведеному нижче списку показано декілька способів навігації кодом, який може зберігатися у великій кількості окремих файлів:

- Як ми вже бачили з розділу щодо [навігації кодом](#), ви можете отримати панель з поясненнями щодо окремого простору назв, класу, функції або змінної наведенням вказівника миші на відповідний запис у коді або утримуванням натиснутою клавіші **Alt**. Ось приклад:



Натискання посилань на оголошення символу або розгортання списку використань надає вам змогу перейти до відповідних місць коду. Подібного ефекту можна досягти за допомогою панелі перегляду **Перегляд коду**, яку ми вже обговорювали.

- **Швидшим способом** переходу до оголошення символу без натискання посилань у підказці є тимчасове вмикання **Режиму навігації кодом** утримуванням натиснутою клавіші **Alt** або **Ctrl**. У цьому режимі можна безпосередньо натиснути будь-який символ у редакторі для переходу до його оголошення.
- **Швидке відкриття**: дуже потужним способом пересування файлами у KDevelop є різноманітні інструменти *швидкого відкриття*. Передбачено чотири різновиди таких інструментів:
 - **Швидке відкриття класу (Навігація → Швидко відкрити клас або Alt-Ctrl-C)**: вам буде показано список всіх класів у поточному сеансі. Почніть вводити (частину) назви класу, і середовище скоротить список до назв, які відповідають введеній частині. Коли список стане достатньо коротким, просто виберіть у ньому елемент за допомогою клавіш зі стрілками вгору і вниз, а KDevelop відкриє місце у кодї, де клас було оголошено.
 - **Швидке відкриття функції (Навігація → Швидко відкрити функцію або Alt-Ctrl-M)**: вам буде показано список всіх (вбудованих) функцій, які є частиною проєктів у поточному сеансі, вибрати потрібну вам функцію можна у той самий спосіб, у який вибирається клас. Зауважте, що у списку можуть бути одразу пункти оголошення та визначення функцій.
 - **Швидке відкриття файла (Навігація → Швидко відкрити файл або Alt-Ctrl-O)**: вам буде показано список всіх файлів, які є частиною проєктів поточного сеансу, вибрати потрібний можна буде у описаний вище спосіб.
 - **Універсальне швидке відкриття (Навігація → Швидко відкрити або Alt-Ctrl-Q)**: якщо ви забули, які комбінації клавіш пов'язано з описаними вище командами, це універсальний спосіб: вам буде показано список всіх файлів, функцій, класів та інших частин проєктів, з якого ви зможете вибрати потрібний вам пункт.
- **Перехід до оголошення або визначення**: під час реалізації функції часто виникає потреба у переході до місця, де цю функцію оголошено, наприклад, для синхронізації списку аргументів між оголошенням і визначенням або для оновлення документації. Виконати

такий перехід можна розташуванням курсора на назві функції з наступним вибором пункту меню **Навігація** → **Перейти до оголошення** (або натисканням комбінації клавіш **Ctrl-**). Повернутися до визначення можна у декілька способів:

- За допомогою пункту меню **Навігація** → **Перейти до визначення** (або натискання комбінації клавіш **Ctrl-**).
- За допомогою пункту меню **Навігація** → **Попередній переглянути контекст** (або натискання комбінації клавіш **Meta-←**), як це описано далі.

ПРИМІТКА

Перехід до оголошення символу не обмежується лише функціями, які ви реалізуєте. Цей спосіб працює і у разі розташування курсора на назві (локальної, загальної чи вбудованої) змінної: вибір відповідного пункту меню призведе до пересування області перегляду до оголошення символу. Отже ви можете, наприклад, розташувати курсор на назві класу у оголошенні змінної функції і перейти до оголошення цього класу.

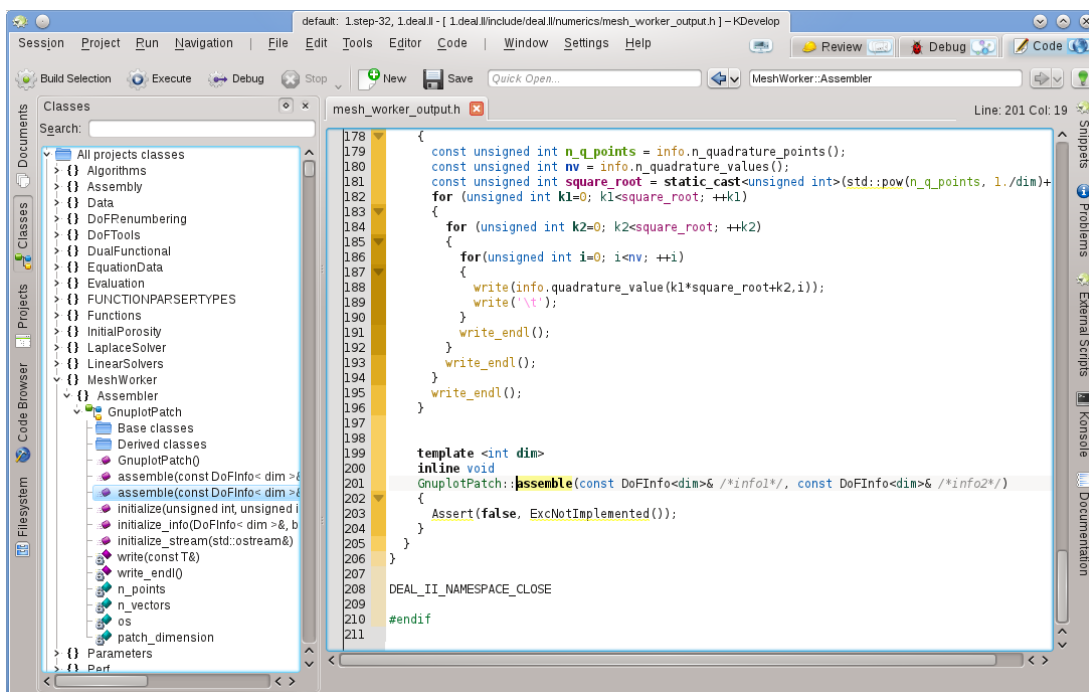
- **Перемкнути визначення/оголошення:** у наведеному вище прикладі для переходу до місця оголошення поточної функції нам потрібно було спочатку розташувати курсор на назві функції. Якщо вам не хочеться цього робити, скористайтеся пунктом меню **Навігація** → **Перемкнути визначення/оголошення** (або натисніть комбінацію клавіш **Shift-Ctrl-C**), щоб перейти до оголошення функції, у якій перебуває курсор. Повторний вибір цього пункту меню поверне курсор назад до визначення функції.
- **Попереднє/ Наступне використання:** розташування курсора на назві локальної змінної з наступним вибором пункту меню **Навігація** → **Наступне використання** (або натискання комбінації клавіш **Meta-Shift-→**) переведе вас до наступного використання цієї змінної у кодї. Зауважте, що буде виконано не лише пошук наступного використання назви змінної, але і взято до уваги і те, що змінні з тією самою назвою, але у інших просторах назв, є іншими. Те саме стосується і назв функцій. Вибір пункту **Навігація** → **Попереднє використання** (або натискання комбінації клавіш **Meta-Shift-←**) переведе перегляд до попереднього використання символу.

ПРИМІТКА

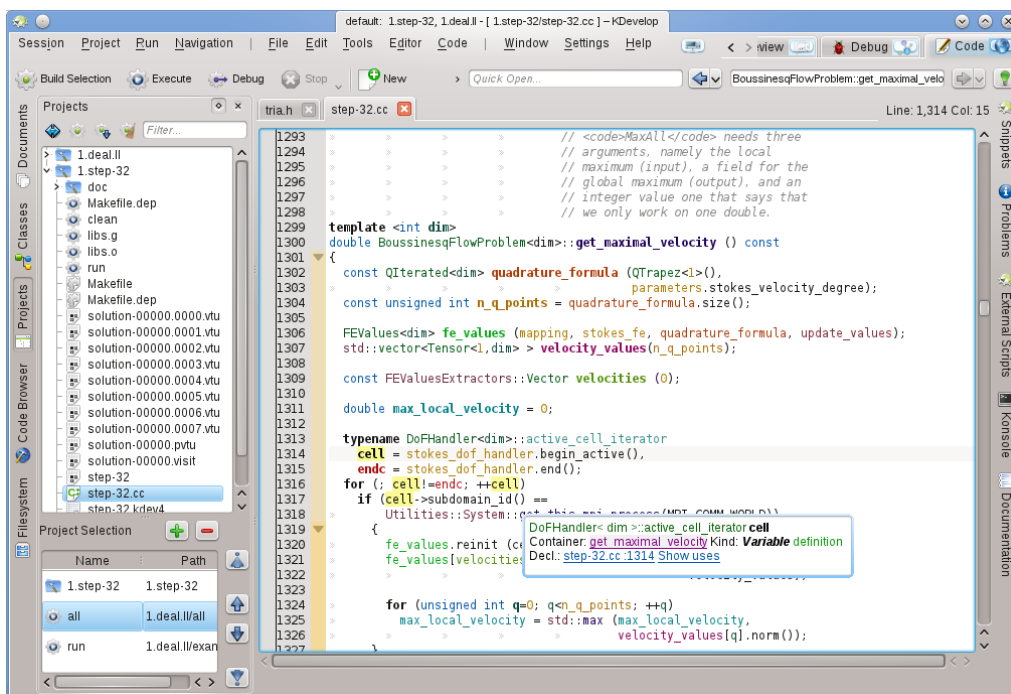
Щоб переглянути список всіх використань назв, якими здійснюватиметься циклічний перехід внаслідок використання цієї команди, розташуйте курсор на назві символу і відкритий панель **Перегляд коду** або натисніть і утримуйте клавішу **Alt**. Докладніше про використання цього прийому можна дізнатися з розділу щодо [навігації кодом](#).

- **Список контекстів:** у переглядачах сторінок інтернету передбачено цю можливість, — ви можете пересуватися вперед і назад списком нещодавно відвіданих сторінок. У KDevelop теж є така можливість, але замість сторінок ви пересуваєтеся *контекстами*. Контекст — це поточне розташування курсора, перехід до якого або з якого було здійснено за допомогою будь-яких дій, окрім натискання клавіші зі стрілками на клавіатурі (наприклад, натискання пункту на панелі підказки, панелі **Перегляд коду**, одного з пунктів меню **Навігація** або використання будь-якої іншої команди навігації). За допомогою пунктів меню **Навігація** → **Попередній переглянути контекст** (**Meta-←**) та **Навігація** → **Наступний переглянутий контекст** (**Meta-→**) ви можете пересуватися списком відвіданих контекстів подібно до того, як за допомогою пунктів **назад** і **вперед** ви можете пересуватися між сторінками, відвіданими за допомогою програми для перегляду інтернету.
- Нарешті, ви можете скористатися панелями інструментів, за допомогою яких можна переходити до різних місць у кодї. Наприклад, за допомогою інструмента **Класи** ви можете користуватися списком всіх просторів назв та класів у всіх проектах поточного сеансу. Ви можете розгортати пункти списку для перегляду списків вбудованих функцій та змінних кожного з класів:

Підручник з KDevelop



Подвійне клацання на пункті (або використання відповідного пункту контекстного меню, яке можна відкрити клацанням правою кнопкою миші) надасть вам змогу перейти до місця оголошення пункту. На інших панелях інструментів можна скористатися подібними ж прийомами, наприклад, на панелі **Проекти** буде наведено список частин сеансу:



Знову ж таки, подвійне клацання на пункті файла призведе до його відкриття.

3.4 Введення коду

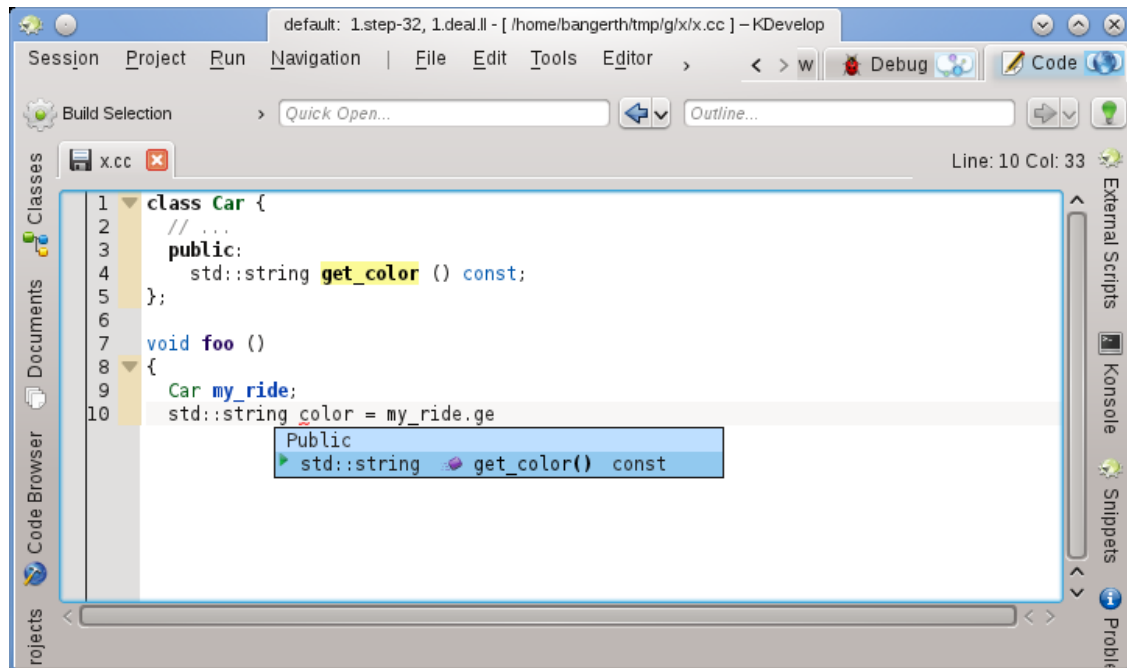
Оскільки KDevelop «розуміє» код ваших проєктів, це середовище може допомогти вам у написанні коду. Нижче наведено огляд деяких з цих допоміжних можливостей.

3.4.1 Автозавершення

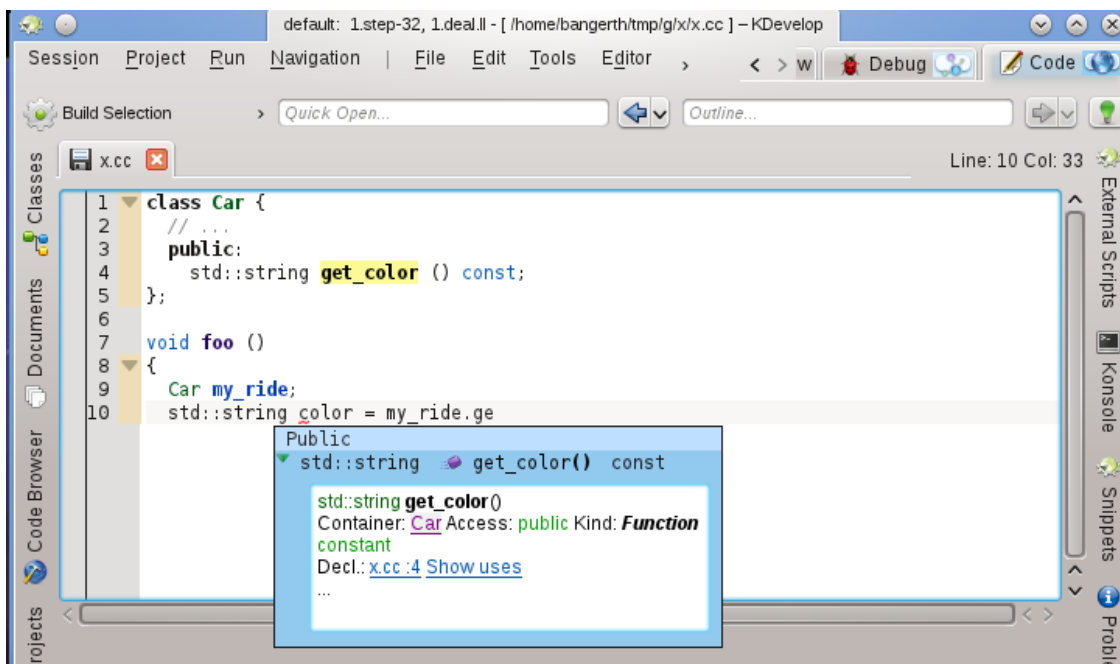
Ймовірно, найкориснішою можливістю є можливість з автоматичного завершення фрагментів коду. Розглянемо, наприклад, такий фрагмент коду:

```
class Car {
    // ...
    public:
        std::string get_color () const;
};
void foo()
{
    Car my_ride;
    // ...якись дії з цією змінною...
    std::string color = my_ride.ge
```

У останньому рядку KDevelop запам'ятає, що змінна `my_ride` належить до типу `Car` і буде автоматично пропонувати доповнення назви вбудованої функції `ge` у форматі `get_color`. Фактично, вам достатньо вводити назву, аж доки список знайдених відповідників не починатиметься з потрібного вам виразу, а потім натиснути клавішу **Enter**:



Зауважте, що ви можете клацнути на панелі підказки, щоб отримати більше даних щодо функції, окрім типу даних, які вона повертає, і те, чи є ця функція відкритою (`public`):



Автоматичне доповнення може значно скоротити витрату часу на введення даних, якщо у вашому проєкті використано змінні або функції з довгими назвами. Крім того, автоматичне доповнення допомагає запобігти друкарським помилкам у назвах (а отже помилкам під час збирання), за його допомогою простіше запам'ятати точні назви функцій. Наприклад, якщо назви всіх ваших функцій отримання даних починаються з `get_`, можливість автоматичного доповнення надасть вам змогу скористатися повним списком таких функцій, щойно ви наберете перші чотири літери назви, що, звичайно, допоможе вам згадати, яку з них ви мали намір використати. Зауважте, що для роботи з автоматичним доповненням не обов'язково, щоб оголошення класу `Car` та змінної `my_ride` перебували у одному файлі, над яким ви зараз працюєте. KDevelop просто має знати, що ці клас і змінну пов'язано, тобто файли, у яких зберігається код цих елементів програми, мають бути частиною проєкту, над яким ви працюєте.

ПРИМІТКА

KDevelop не завжди може здогадатися, коли слід допомогти вам автоматичним доповненням коду. Якщо підказку автоматичного доповнення не було відкрито автоматично, натисніть комбінацію клавіш **Ctrl-Пробіл**, щоб відкрити список варіантів вручну. Загалом же, для того, щоб можна було скористатися автоматичним доповненням, KDevelop має обробити файли коду вашого проєкту. Цю дію буде виконано у фоновому режимі для всіх файлів, які є частиною проєктів поточного сеансу після запуску KDevelop, а також після того, як ви припините вводити дані на частку секунди (тривалість паузи можна змінити).

ПРИМІТКА

KDevelop виконує обробку лише тих файлів, які є файлами коду програми, відповідно до типу MIME файла. Цей тип не встановлюється до першого збереження файла. Отже створення нового файла і введення коду до цього файла не увімкне автоматичної обробки файла для використання даних з метою автоматичного доповнення. Щоб увімкнути обробку, файл доведеться зберегти.

ПРИМІТКА

Для того, щоб можна було скористатися автоматичним доповненням, KDevelop повинен мати доступ до файлів заголовків. Середовище виконає пошук цих файлів у декількох типових теках. Якщо файл заголовка не вдасться знайти автоматично, його пункт у списку буде підкреслено червоною лінією. У такому разі вам слід клацнути на цьому пункті правою кнопкою миші і повідомити KDevelop явним чином, де слід шукати цей файл та дані, які у ньому зберігаються.

ПРИМІТКА

Налаштування автоматичного доповнення обговорено у [іншому розділі цього підручника](#).

3.4.2 Додавання нових класів та реалізація вкладених функцій

У KDevelop передбачено допоміжну програму для додавання нових класів. Її роботу описано у розділі [Створення класу](#). Простий клас C++ можна створити за допомогою основного шаблону C++ з категорії Клас. У допоміжній програмі ви можете вибрати попередні функції-елементи, наприклад порожній конструктор, конструктор копіювання та деструктор.

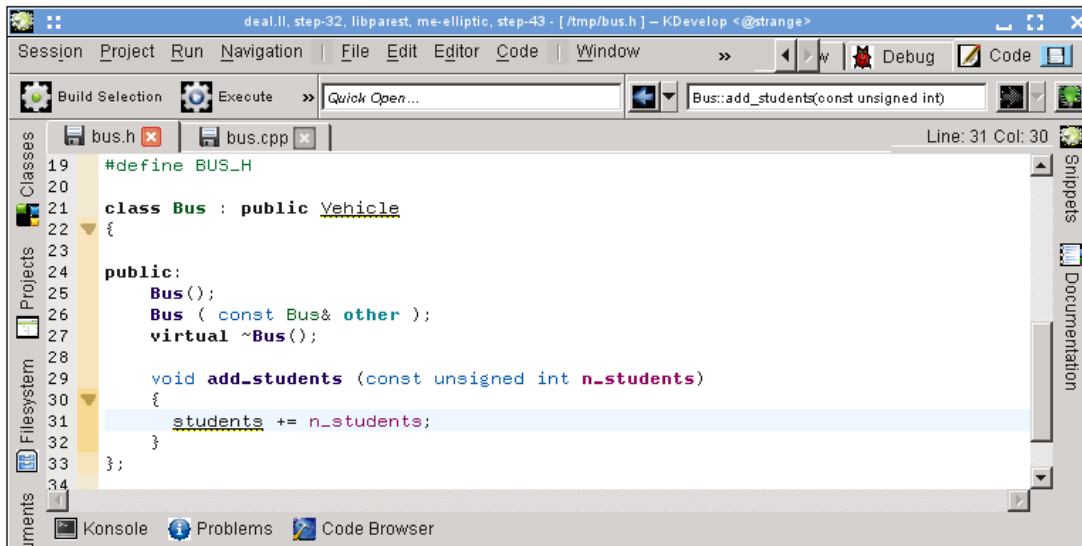
Після завершення роботи допоміжної програми буде створено і відкрито у редакторі нові файли. Основу файла заголовків вже буде створено, а у новому класі є всі вибрані нами вбудовані функції. Наступними двома кроками мають бути документування класу та його вбудованих функцій та реалізація цих класів і функцій. Нижче ми обговоримо допоміжні засоби з документування класів і функцій. Щоб реалізувати спеціальні функції, які ми вже додали, просто перейдіть на вкладку **bus.cpp**, де вже є каркас наших функцій:

```

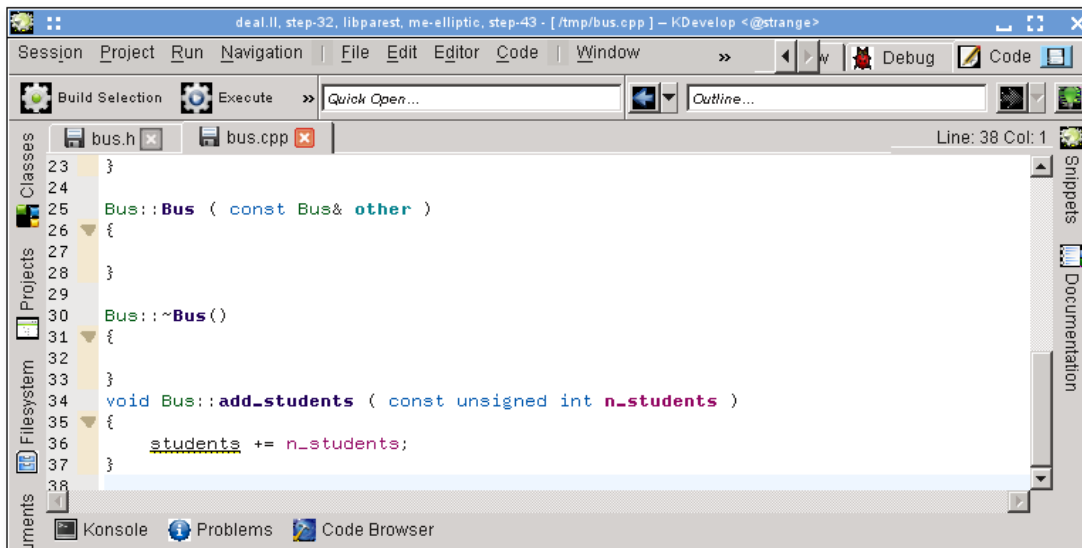
1  /*
2  Copyright 2011 <copyright holder> <email>
3
4  Licensed under the Apache License, Version 2.0 (the "License");
5  you may not use this file except in compliance with the License.
6  You may obtain a copy of the License at
7
8  http://www.apache.org/licenses/LICENSE-2.0
9
10 Unless required by applicable law or agreed to in writing, software
11 distributed under the License is distributed on an "AS IS" BASIS,
12 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 See the license for the specific language governing permissions and
14 limitations under the License.
15 */
16
17
18 #include "bus.h"
19
20 Bus::Bus()
21 {
22 }
23
24 Bus::Bus ( const Bus& other )
25 {
26 }
27
28 }
29
30 Bus::~Bus()
31 {
32 }
33

```

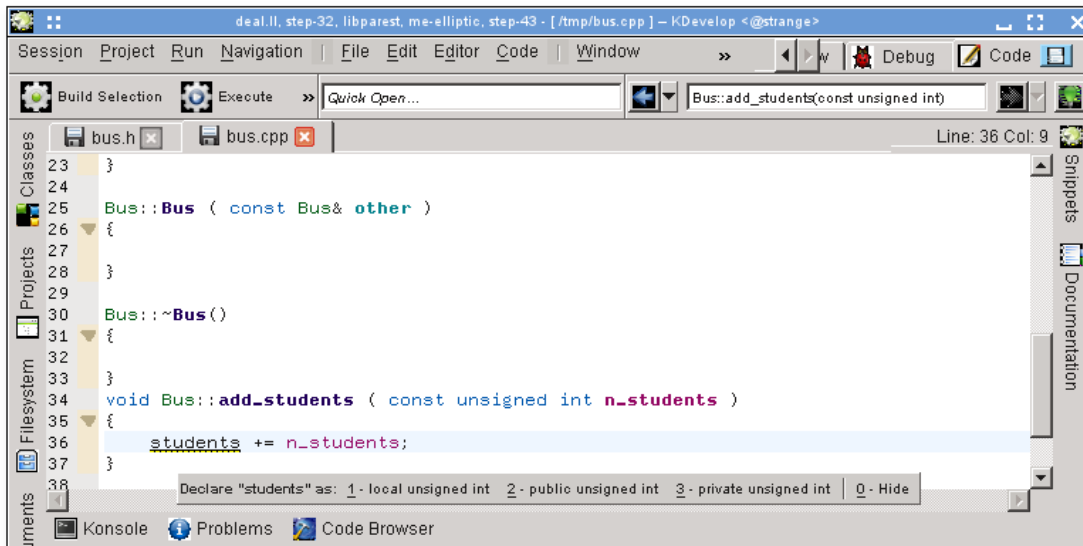
Щоб додати нові вкладені функції, поверніться на вкладку **bus.h** і додайте назву функції. Наприклад, можна додати таке:



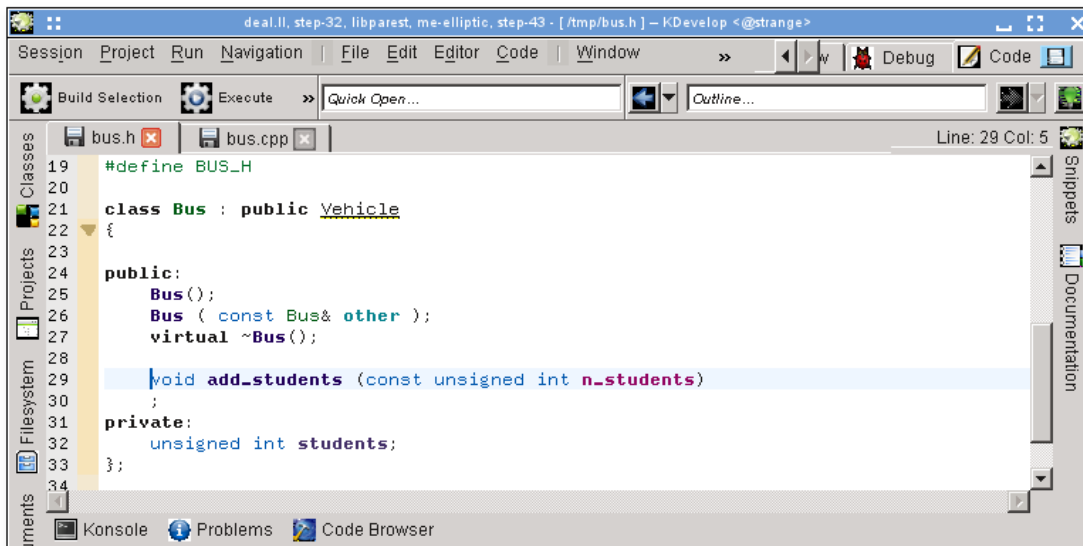
Зауважте, що реалізацію вже розпочато. Але у межах багатьох стилів програмування функції не слід реалізовувати у файлах заголовків, це слід робити у відповідних файлах `.cpp`. Для цього розташуйте курсор на назві функції і скористайтеся пунктом меню **Код** → **Пересунути до коду** або натисніть комбінацію клавіш **Ctrl-Alt-S**. Код між фігурними дужками буде пересунуто з файла заголовків (його буде замінено на крапку з комою для завершення оголошення функції) до відповідного файла коду:



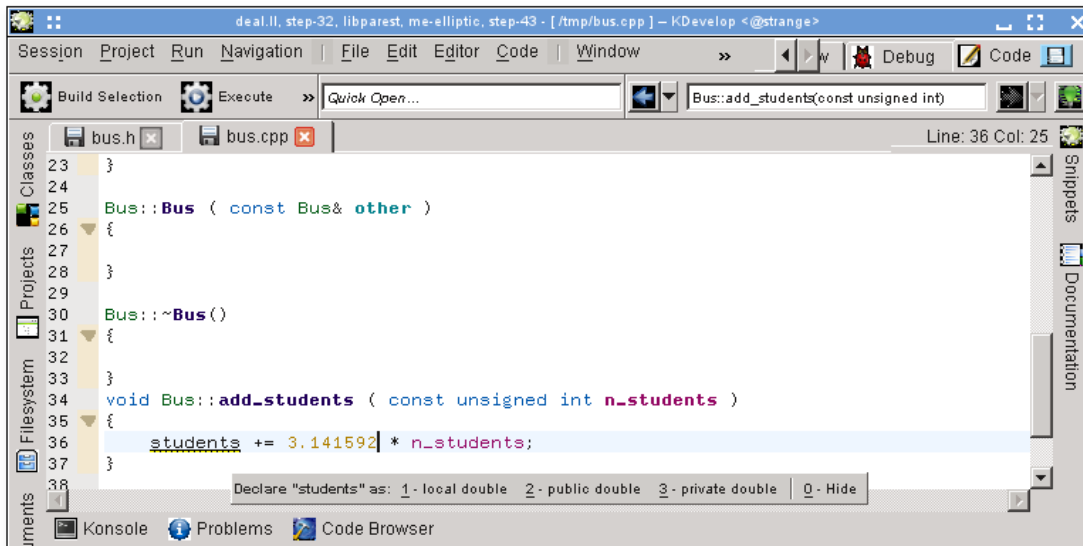
Введення коду розпочато. Змінна `students` має, ймовірно, бути вбудованою змінною класу `Bus`, але її ще не додано до цього класу. KDevelop підкреслено цю змінну, щоб позначити те, що середовищу ще нічого не відомо про цю змінну. Цю проблему просто розв'язати: натискання назви змінної відкриє таку панель підказки:



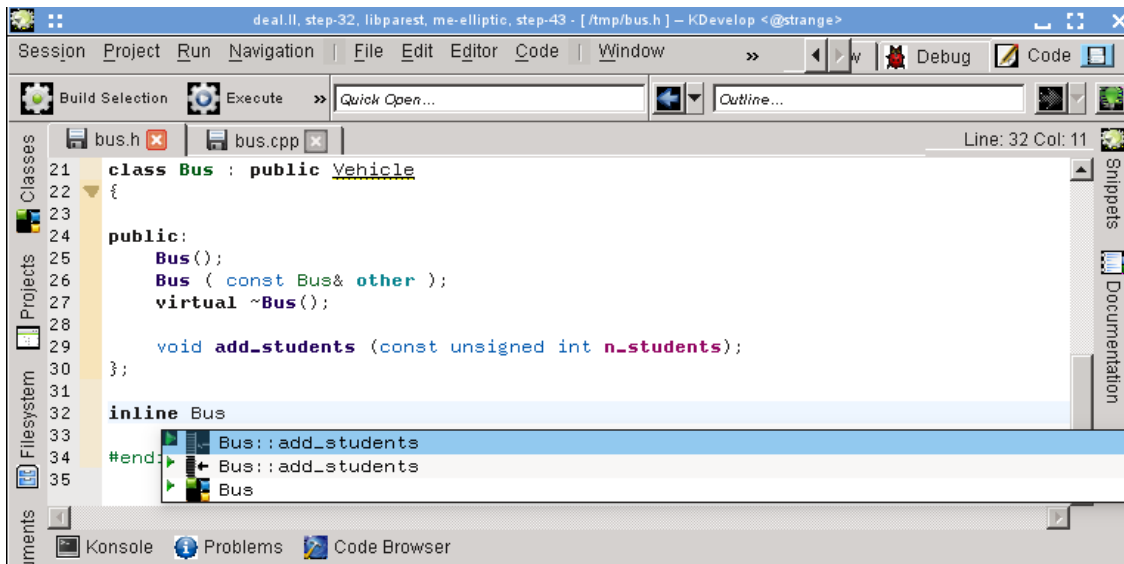
(Того самого результату можна досягти за допомогою пункту контекстного меню **Розв'язати: оголосити як**.) Давайте виберемо «3 - private unsigned int» (за допомогою миші або натискання комбінації клавіш **Alt-3**). Ось що буде додано до файла заголовків:



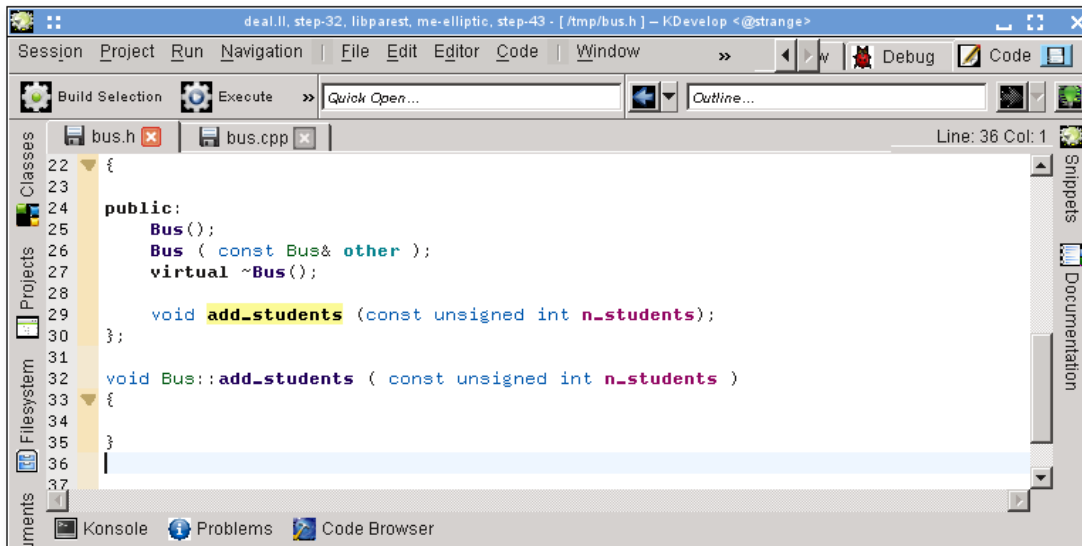
Варто зауважити, що KDevelop визначає тип змінної, яку слід оголосити, за виразом, використаним для її ініціалізації. Наприклад, якщо було використано додавання у такому доволі безсумнівному форматі, середовище запропонує тип змінної `double`:



Нарешті, метод використання пункту меню **Код** → **Пересунути до коду** не завжди призводить до вставлення вбудованої функції до бажаного місця у кодї. Наприклад, вам може бути потрібним визначення коду як `inline` і розташування його у нижній частині файла заголовків. У такому разі вкажіть оголошення і почніть введення визначення функції ось так:



KDevelop автоматично пропонує всіх можливі варіанти доповнення. Вибір одного з двох записів `add_students` призведе до створення такого коду, у якому вже буде заповнено список параметрів:



ПРИМІТКА
 У нашому прикладі, використання одного з варіантів інструмента автоматичного доповнення призводить до додавання належного коду, але, на жаль, вилучає позначку `inline`, яку вже написано. Про цю ваду повідомлено як про ваду [KDevelop №274245](#).

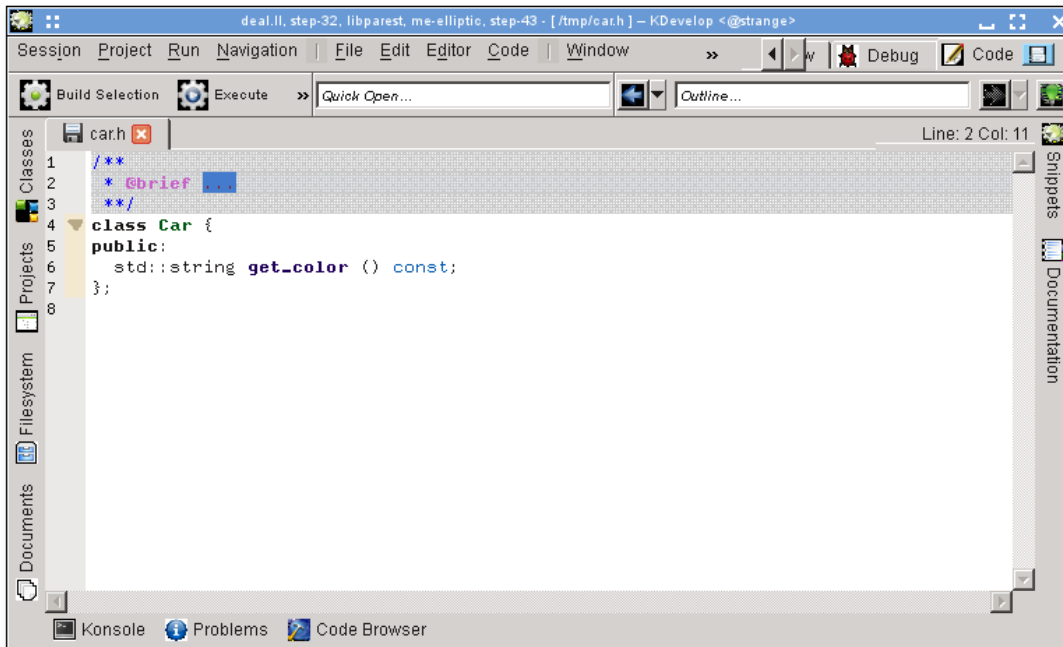
3.4.3 Оголошення для документування

Хороший код завжди добре документовано, як на рівні реалізації алгоритмів у функціях, так і на рівні інтерфейсу, — тобто класи (вбудовані і загальні) функції та (вбудовані і загальні) змінні слід документувати, щоб всім було зрозумілим їхнє призначення, можливі значення аргументів, попередні та остаточні умови тощо. У документуванні інтерфейсів фактичним стандартом є формат коментарів `doxygen`, дані його можна видобути і показати на придатних для пошуку інтернет-сторінках.

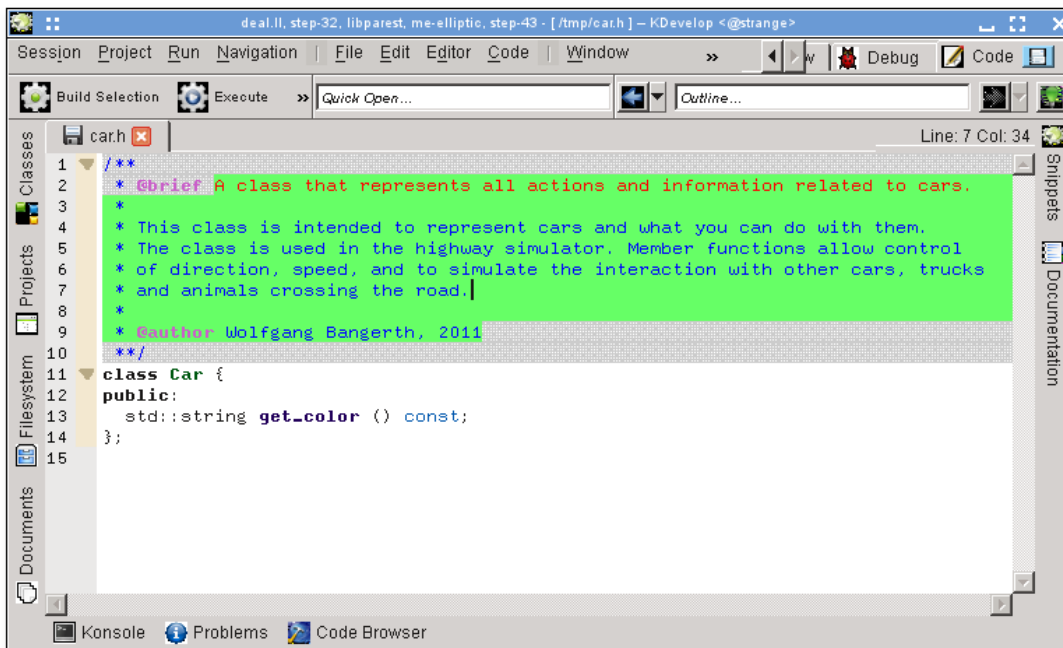
У KDevelop передбачено підтримку коментарів у цьому форматі за допомогою скорочення для створення оболонки-коментаря, який документуватиме клас або вбудовану функцію. Наприклад, припустімо ви вже створили такий код:

```
class Car {
public:
    std::string get_color () const;
};
```

Тепер нехай вам потрібно додати документацію для класу і вбудованої функції. Для цього пересуньте курсор на перший рядок коду і скористайтесь пунктом меню **Код** → **Документувати оголошення** або натисніть комбінацію клавіш **Alt-Shift-D**. KDevelop відкриє таке діалогове вікно:

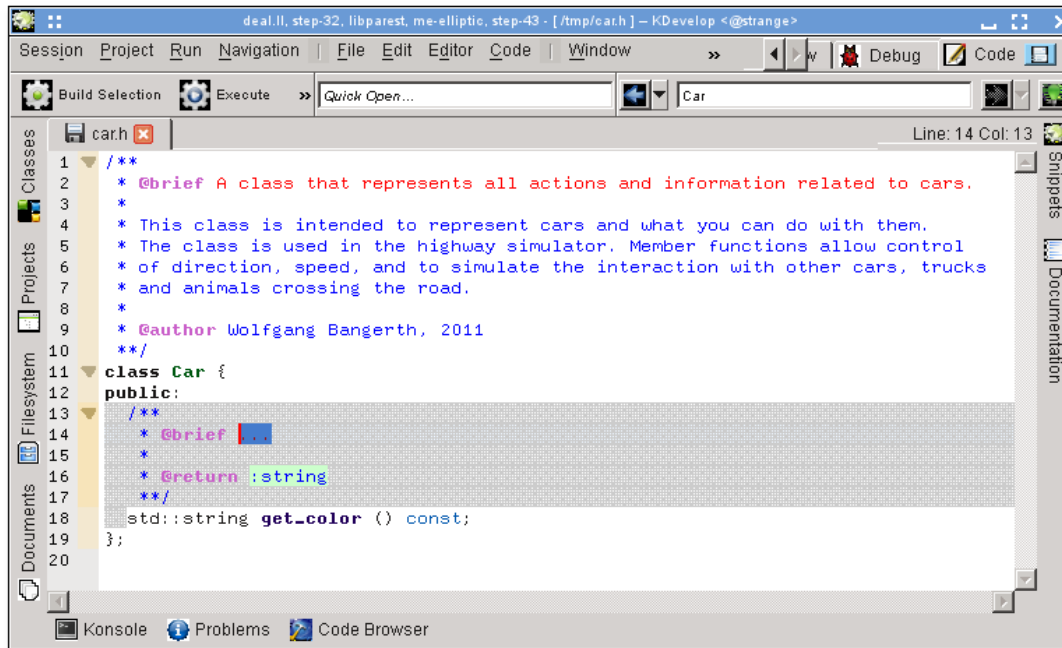


Курсор вже перебуває у затіненій області, щоб ви могли ввести короткий опис (після ключового слова дохуген `@brief`) цього класу. Тепер ви можете продовжити додавати документацію до цього коментаря, який краще пояснюватиме призначення класу:

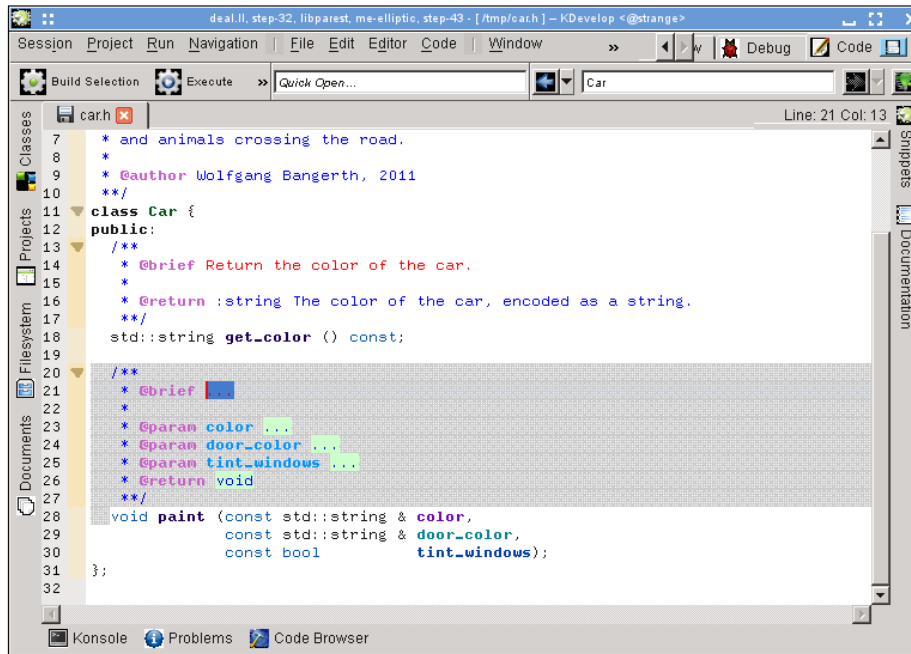


Доки редагування відбуватиметься у коментарі текст коментаря буде позначено зеленим кольором (колір буде знято, шойно ви виведете курсор за межі коментаря). Коли ви наприкінці рядка натиснете клавішу **Enter**, KDevelop автоматично додасть новий рядок, що починатиметься з зірочки та відступу у один символ.

Тепер виконаємо документування вбудованої функції. Знову розташуйте курсор на рядку оголошення і виберіть пункт меню **Код** → **Документувати оголошення** або натисніть комбінацію клавіш **Alt-Shift-D**:



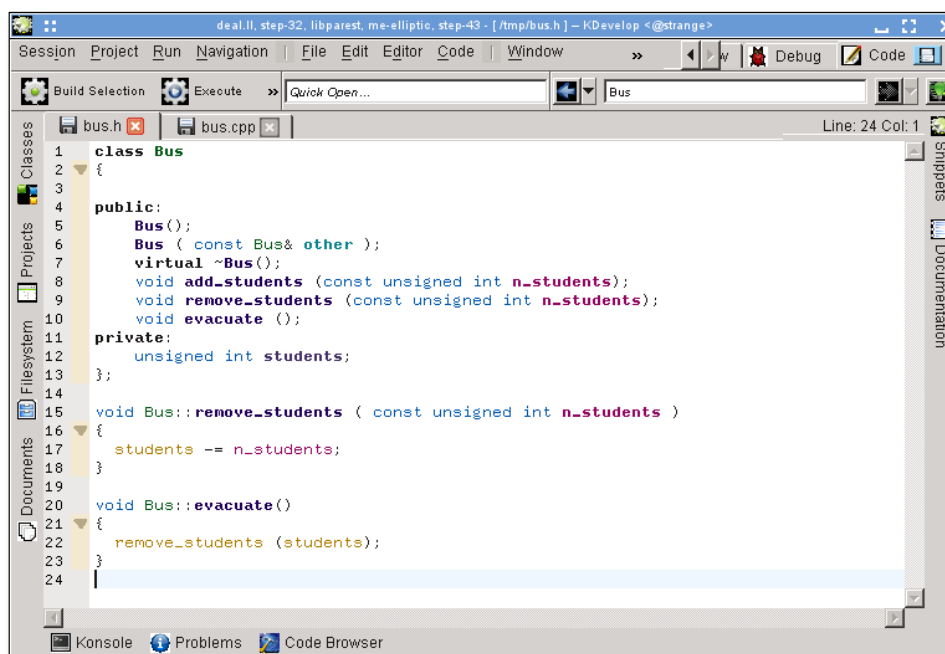
Знову ж таки, KDevelop автоматично створить каркас коментаря разом з документації щодо самої функції, а також типом даних, які вона повертає. У нашому випадку назва функції доволі очевидно описує її призначення, але часто аргументи функції можуть бути доволі неочевидними, їх слід документувати окремо. Наприклад, розгляньмо трохи цікавішу функцію та коментар, який автоматично створить для неї KDevelop:



У нашому прикладі у запропонованому коментарі вже містяться всі поля Doxygen для окремих параметрів.

3.4.4 Перейменування змінних, функцій і класів

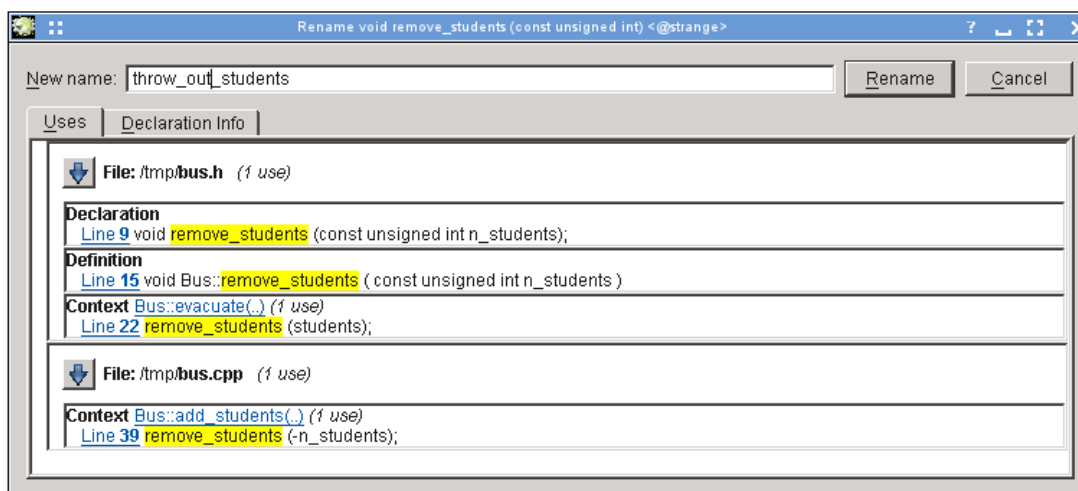
Іноді виникає потреба у перейменуванні функції, класу або змінної. Наприклад, нехай маємо такий код:



Нехай пізніше ми зрозуміємо, що назва `remove_students` є невдалою, і її варто назвати, скажімо, `throw_out_students`. Ви могли б виконати пошук з заміною назви функції, але у такого способу є два недоліки:

- Функцію може бути використано у декількох файлах.
- Нам потрібно перейменувати лише цю функцію і не чіпати функції, які можуть мати ту саму назву, але які оголошено у інших класах або просторах назв.

Обидві ці проблеми можна розв'язати наведенням курсора на всі використання назви функції з наступним вибором пункту **Код** → **Перейменувати оголошення** (або клацанням правою кнопкою миші на назві з вибором пункту **Перейменувати Bus::remove_students**). У відповідь буде відкрито діалогове вікно, за допомогою якого ви зможете ввести нову назву функції і переглянути всі місця, де використано функцію:

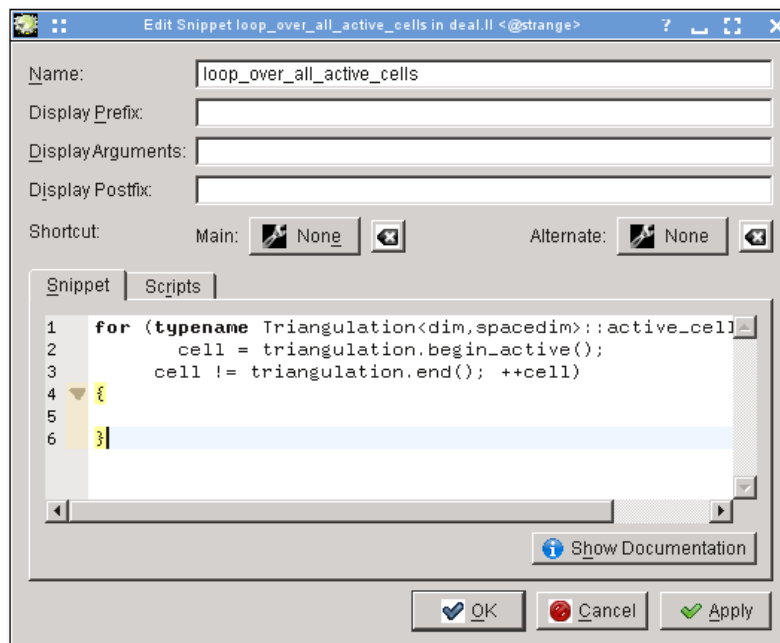


3.4.5 Фрагменти коду

У більшості проектів зустрічаються фрагменти коду, які доводиться використовувати доволі часто. Приклади: інструкції компілятора, цикл для всіх інструкцій, для функцій запису інтерфейсу користувача перевірки введених користувачем даних без відкриття вікна повідомлення про помилку. У проекті автора цього підручника часто використовується такий різновид коду:

```
for (typename Triangulation::active_cell_iterator
     cell = triangulation.begin_active();
     cell != triangulation.end(); ++cell)
    ... якісь дії над cell ...
```

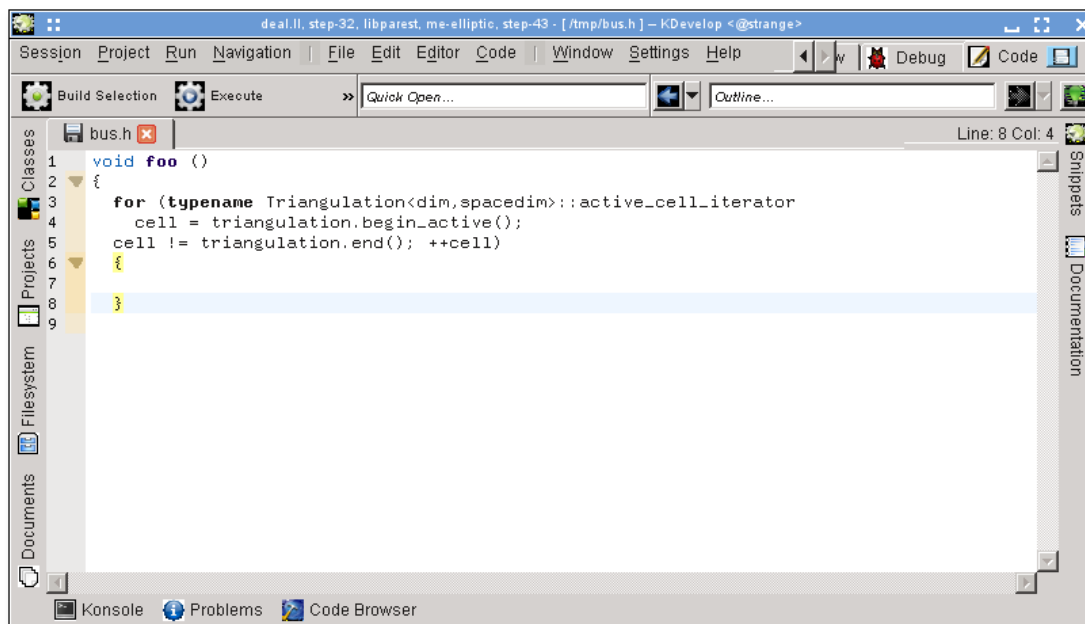
Замість введення таких фрагментів тексту раз за разом, що може призвести до супутніх друкарських помилок ви можете скористатися інструментом **Фрагменти KDevelop**. Для цього відкрийте панель інструментів (див. розділ **Інструменти і панелі перегляду**, якщо відповідної кнопки ще немає на панелях навколо панелі редагування). Натисніть кнопку **Додати сховище** (трохи невдала назва: за допомогою цієї кнопки ви можете створити іменовану збірку фрагментів коду певного типу, наприклад, фрагментів коду C++) і створіть порожнє сховище. Потім натисніть кнопку **+**, щоб додати фрагмент і відкрити таке діалогове вікно:



ПРИМІТКА

У назві фрагмента не повинно бути пробілів та інших спеціальних символів, оскільки такий фрагмент повинен мати назву, подібну до назви звичайної функції або змінної (причини цього викладено нижче).

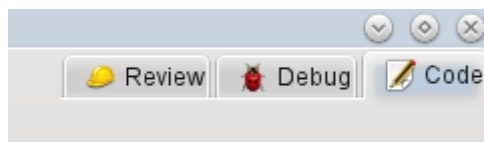
Щоб скористатися таким чином визначеним фрагментом, вам слід просто почати вводити у кодї його назву, як ви це робите для будь-яких інших функцій або змінних. Для введених літер буде відкрито панель автоматичного доповнення (це означає, що можна використовувати будь-які достатньо довгі для розрізнення назви фрагментів, зокрема назви, подібні до використаної нами), якщо ви виберете відповідний пункт фрагмента на панелі підказки автоматичного доповнення (наприклад, натисканням клавіші **Enter**), вже введenu частину назви фрагмента буде замінено повним текстом фрагмента з відповідними відступами:



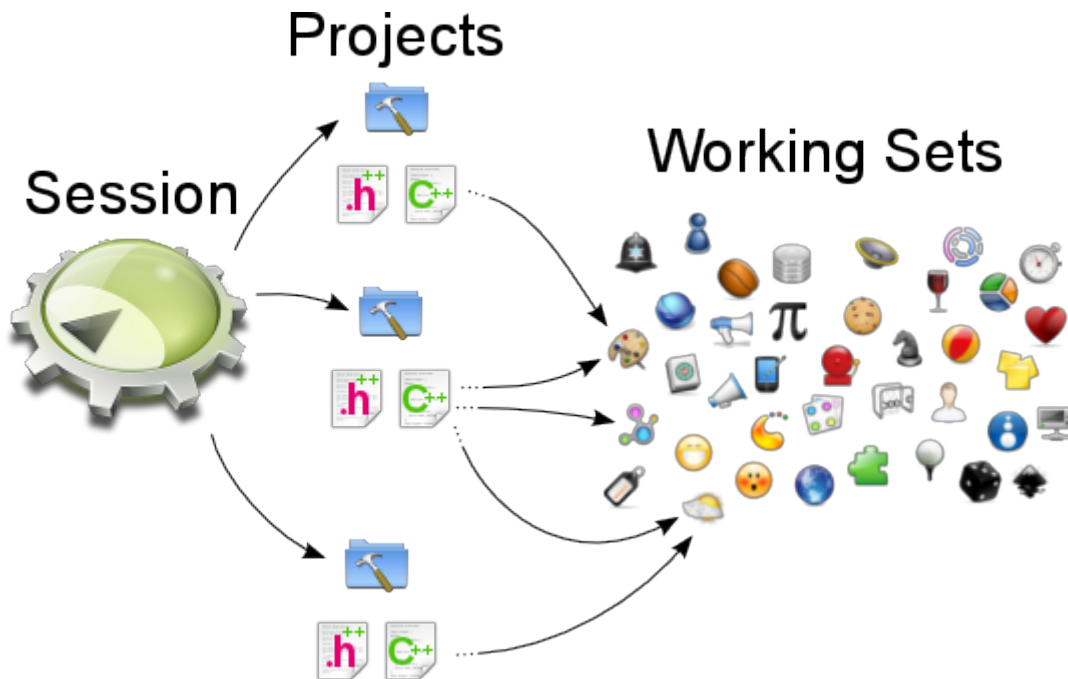
Зауважте, що для користування фрагментами не потрібно тримати відкритою і видимою панель інструмента **Фрагменти**: цією панельлю слід користуватися, лише для визначення нових фрагментів. Іншим, але менш зручним, способом перегляду фрагментів є просте натискання його пункту на відповідній панелі інструмента.

ПРИМІТКА
 Фрагменти є набагато потужнішими, ніж було пояснено. Повний опис можливостей, які вони надають, ознайомтеся з [докладною документацією з інструмента «Фрагменти»](#).

3.5 Режими і робочі набори

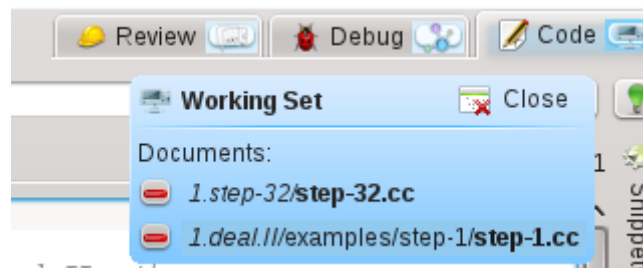


Якщо вже ви дочитали до цього місця, зверніть увагу на верхню праву частину головного вікна KDevelop: як можна бачити з наведеного знімка, KDevelop може працювати у трьох **режимах**: **Код** (роботу у цьому режимі ми обговорювали у поточному розділі, коли оглядали роботу з кодом), **Налагоджування** (див. [Налагоджування програм](#)) та **Перегляд** (див. [Робота з системами керування версіями](#)).



Для кожного з режимів передбачено власний набір інструментів, кнопки яких розташовано вздовж периметра вікна середовища, для кожного режиму передбачено власний *робочий набір* відкритих файлів і документів. Крім того, кожен такий робочий набір пов'язано з поточним сеансом, тобто маємо взаємозв'язок, подібний до показаного вище. Зауважте, що файли у робочому наборі є частиною одного сеансу, але можуть належати до різних проектів, які є частинами цього сеансу.

Якщо ви відкрили вікно KDevelop вперше, робочий набір буде порожнім, — це не відкрито жодного файла. Але з відкриттям файлів для редагування (або налагоджування, перегляду у інших режимах) ваш робочий набір розширюватиметься. Той факт, що ваш робочий набір не є порожнім, буде показано символом на вкладці, подібно до символу на знімку, наведеному нижче. Кожного разу, коли ви завершуватимете роботу KDevelop, а потім починатимете її знову, робочий набір буде зберігатися і відновлюватися, тобто ви працюватимете з одним набором відкритих файлів.



Якщо ви наведете вказівник миші на символ робочого набору, середовище покаже вам список файлів, які відкрито у робочому наборі (у нашому прикладі список з файлів `step-32.c` та `step-1.cc`). Натискання кнопки з червоним мінусом призведе до закриття вкладки відповідного файла. Натискання кнопки з відповідною назвою надасть вам змогу **закрити** весь робочий набір одразу (тобто закрити всі відкриті файли). Сенс цього пункту у тому, що за його допомогою можна не лише закрити всі файли, але і зберегти робочий набір, а також відкрити новий порожній файл. Ось як це виглядає:



Зверніть увагу на дві піктограми ліворуч від трьох заголовків вкладок режимів (піктограму з сердечком та іншу піктограму лівіше). Кожній з цих піктограм відповідає збережений робочий набір, окрім вже відкритого робочого набору. Якщо ви наведете вказівник миші на піктограму з сердечком, ви побачите щось таке:



Середовище показує, що поточний робочий набір складається з двох файлів, та демонструє їхні назви: `Makefile` та `changes.h`. Натискання кнопки **Завантажити** призведе до закриття і збереження поточного робочого набору (який у нашому прикладі складається з файлів `tria.h` і `tria.cc`), після чого буде відкрито вибраний робочий набір. Крім того, ви можете назавжди вилучити робочий набір, тобто усунути його зі списку збережених робочих наборів.

3.6 Деякі корисні клавіатурні скорочення

У редакторі KDevelop використано всі звичайні клавіатурні скорочення для дій з редагування. Крім того, у редакторі передбачено декілька додаткових дій, пов'язаних з редагуванням коду програм. Деякі з цих дій мають відповідні типові клавіатурні скорочення. Нижче наведено частину з корисних клавіатурних скорочень для дій:

Пересування кодом	
Ctrl-Alt-O	Пришвидшене відкриття файла: вкажіть частину назви файла і виберіть той з них у каталозі проектів поточного сеансу, який вам потрібен; цей файл і буде відкрито.
Ctrl-Alt-C	Пришвидшене відкриття класу: вкажіть частину назви класу і виберіть ту з назв класів, яка вам потрібна; курсор буде переведено до рядка оголошення класу.
Ctrl-Alt-M	Пришвидшене відкриття функції: вкажіть частину назви функції (частини класу) і виберіть з запропонованих варіантів той, який вам потрібен; зауважте, що у списку буде показано оголошення і визначення, а курсор буде переведено до вибраного вами пункту.
Ctrl-Alt-Q	Універсальне пришвидшення відкриття: вкажіть будь-які дані (назву файла, назву класу, назву функції), і вам буде показано список всіх частин проекту, які відповідають критерію пошуку.
Ctrl-Alt-N	Огляд: показує список всіх дій, які виконуються у файлі, наприклад, оголошень класів та визначень функцій.

Ctrl-,	Перейти до визначення функції, якщо курсор перебуває у місці її оголошення.
Ctrl-.	Перейти до оголошення функції або змінної, якщо курсор перебуває у позиції визначення функції.
Ctrl-Alt-PageDown	Перейти до наступної функції
Ctrl-Alt-PageUp	Перейти до попередньої функції
Ctrl-G	Перейти до рядка

Пошук і заміна	
Ctrl-F	Знайти
F3	Знайти далі
Ctrl-R	Замінити
Ctrl-Alt-F	Знайти і замінити у декількох файлах

Інші команди	
Ctrl-<u>_</u>	Згорнути один рівень: вилучити блок з перегляду, наприклад, якщо ви бажаєте зосередитися на загальнішій картині у функції
Ctrl-+	Розгорнути один рівень: скасувати згортання.
Ctrl-D	Зняти коментування з позначеного фрагмента тексту або поточного рядка.
Ctrl-Shift-D	Додати позначки коментаря до позначеного тексту або поточного рядка.
Alt-Shift-D	Документувати поточну функцію. Якщо курсор перебуває на оголошенні функції або класу, натискання цієї комбінації клавіш призведе до створення коментаря у форматі doxygen зі списком параметрів, значень, які повертаються, тощо.
Ctrl-T	Поміняти місцями поточний і попередній символи
Ctrl-K	Вилучити поточний рядок (зауваження: ця дія не збігається за результатом з дією «вилучити звідси до кінця рядка» у emacs).

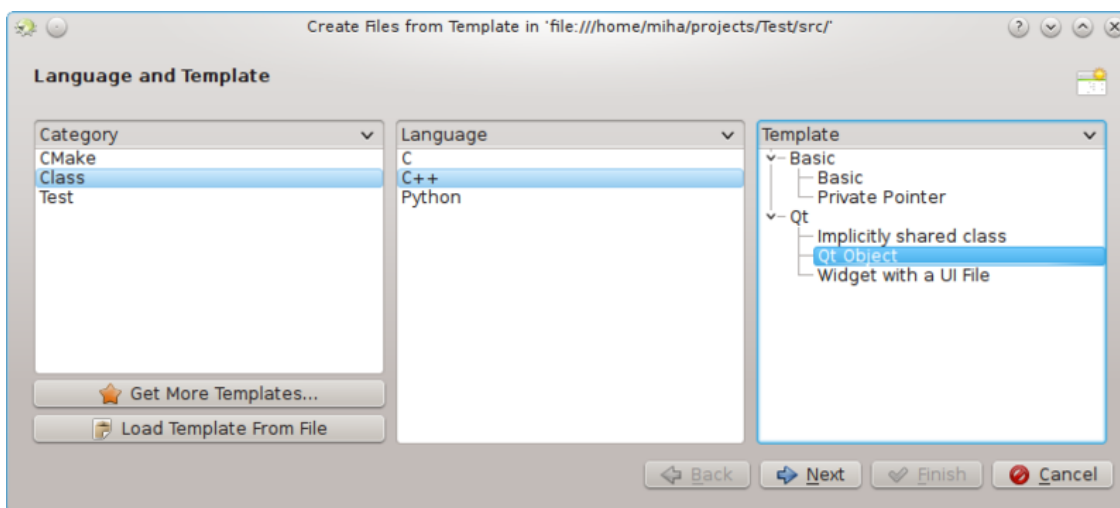
Розділ 4

Створення коду за допомогою шаблонів

У KDevelop для створення початкового коду можна використовувати шаблони. Отже, за допомогою шаблонів ви можете уникнути повторного написання поширених частин коду.

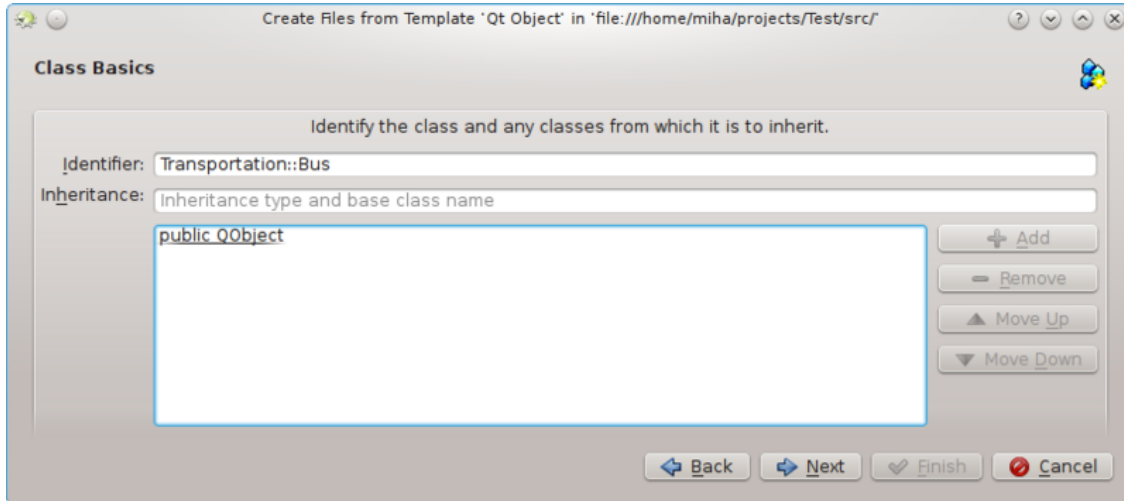
4.1 Створення класу

Найпоширенішим випадком використання створення коду за шаблоном є, ймовірно, написання нового класу. Щоб створити новий клас у вже створеному проекті, наведіть вказівник миші на теку проекту, клацніть правою кнопкою миші і виберіть у контекстному меню пункт **Створити з шаблону**. Те саме діалогове вікно може бути відкрито за допомогою меню, пункту **Файл** → **Створити з шаблону**, але використання теки проекту має перевагу встановлення базової адреси для вихідних файлів. Виберіть **Клас** на панелі вибору категорії, бажану мову програмування та шаблон. Після вибору шаблону класу вам слід вказати інші характеристики нового класу.



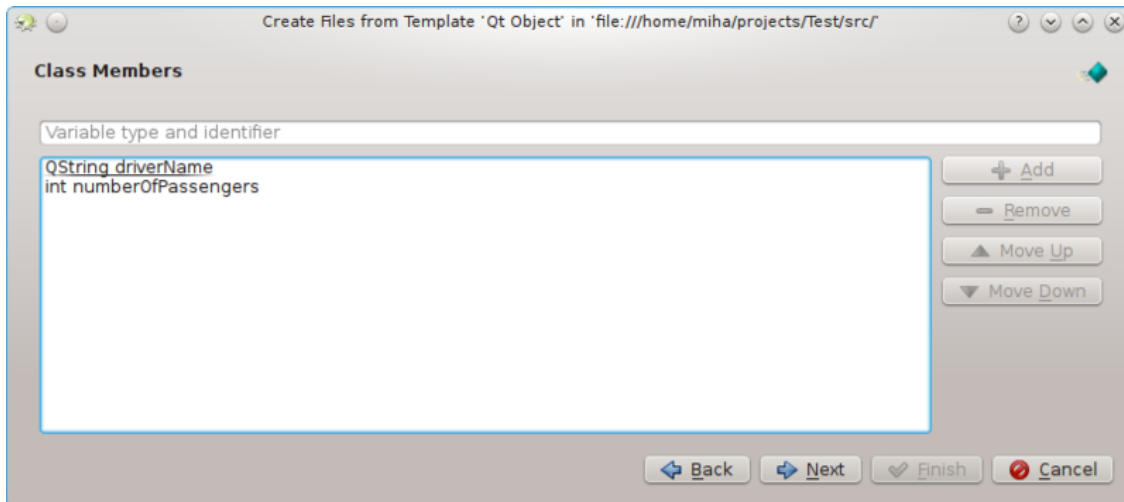
Спочатку слід вказати ідентифікатор нового класу. Це може бути проста назва (наприклад `Bus`) або повний ідентифікатор у просторі назв (наприклад `Transportation::Bus`). У останньому випадку KDevelop виконає обробку ідентифікатора і відповідним чином відокремить простір назв від назви класу. За допомогою цієї самої сторінки ви можете додати основні класи для нового класу. Ви можете зауважити, що у деяких шаблонах уже передбачено вибір основних класів. Ви можете просто вилучити зайві і/або додати інші основні класи. Тут вам слід вказати інструкцію наслідування повністю. Звичайно ж, така інструкція залежить

від вибраної мови програмування, це буде `public QObject` для C++, `extends SomeClass` для PHP або просто назва класу для Python.



За допомогою наступної сторінки ви можете вибрати віртуальні методи з успадкованих класів, а також типові конструктори, деструктори і оператори. Реалізувати метод у новому класі можна простим позначенням пунктів з підписами методів.

Після натискання кнопки **Далі** буде показано сторінку, за допомогою якої ви можете додати до класу методи. Залежно від вибраного шаблону, методи може бути реалізовано у новому класі як змінні-елементи або може бути створено властивості з функціями встановлення та отримання значень. У мовах, де типи змінних слід оголошувати, зокрема у C++, вам слід вказати одразу тип і назву елемента, наприклад `int number` або `QString name`. У інших мовах можна не вказувати тип, але все ж варто його визначити, оскільки таке визначення може бути корисним для подальшої роботи з шаблоном.



За допомогою наступних сторінок ви зможете вибрати умови ліцензування вашого нового класу, встановити нетипові параметри, характерні для вибраного шаблону, та налаштувати місце зберігання виведених даних для всіх створених програмою файлів. Натискання кнопки **Завершити** призведе до завершення роботи допоміжної програми і створення нового класу. Створені допоміжною програмою файли буде одразу відкрито у вікні редактора, отже ви зможете негайно перейти до додавання потрібного коду.

Після створення класу C++ програма запропонує вам додати клас до певної цілі у проекті. Виберіть у діалоговому вікні відповідну ціль або закрийте діалогове вікно і додайте файли до цілі вручну.

Якщо ви вибрали шаблон **Об'єкт Qt**, вибрали певні типові методи та додали дві змінні-елемента, результат має виглядати так, як це зображено нижче.

```

Bus.h
Bus.cpp

/*
 * This file is licensed under the Free Transportation License 3.14
 */

#ifndef TRANSPORTATION_BUS_H
#define TRANSPORTATION_BUS_H

#include <QtCore/QObject>

namespace Transportation {

class BusPrivate;

class Bus : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString driverName READ driverName WRITE setDriverName)
    Q_PROPERTY(int numberOfPassengers READ numberOfPassengers WRITE setNumberOfPassengers)

public:
    Bus();
    Bus(const Bus& other);
    ~Bus();

    QString driverName() const;
    int numberOfPassengers() const;

public Q_SLOTS:
    void setDriverName(const QString& driverName);
    void setNumberOfPassengers(int numberOfPassengers);

private:
    Q_DECLARE_PRIVATE(Bus)
};
}

#endif // TRANSPORTATION_BUS_H

```

Як можна бачити, дані-елементи перетворено на властивості Qt з функціями доступу та макросами `Q_PROPERTY`. Крім того, аргументи функцій встановлення передано з посиланнями на сталі там, де це потрібно. Крім того, оголошено закритий клас (`private`) та створено закритий вказівник з `Q_DECLARE_PRIVATE`. Все це зроблено відповідним шаблоном. Вибір іншого шаблону на першому кроці може призвести до зовсім іншого результату.

4.2 Створення тесту модуля

Хоча у більшості комплексів для тестування кожен з тестів має бути окремим класом, у KDevelop передбачено метод спрощення створення тестів для модулів програми. Щоб створити новий тест, наведіть вказівник миші на теку проекту, клацніть правою кнопкою миші і виберіть у контекстному меню пункт **Створити з шаблону** [#8230](#). На сторінці вибору шаблону у полі категорії виберіть **Перевірка**, потім виберіть мову програмування і шаблон та натисніть кнопку **Далі**.

Вас попросять ввести назву перевірки і список варіантів перевірки. Для варіантів вам слід вказати список назв. У деяких комплексах для тестування модулів, зокрема PyUnit і RHPUnit, слід назви варіантів перевірки мають починатися з особливого префікса. У KDevelop за додавання префіксів відповідає шаблон, отже вам не потрібно вказувати префікс для варіантів. Після натискання кнопки **Далі** вкажіть умови ліцензування і розташування файлів виведених даних. KDevelop виконає решту роботи у автоматичному режимі.

Створені таким чином тести модулів не буде автоматично додано до жодної з цілей проекту. Якщо ви користуєтесь CTest або якимось іншим комплексом для тестування, не забудьте додати нові файли до цілі.

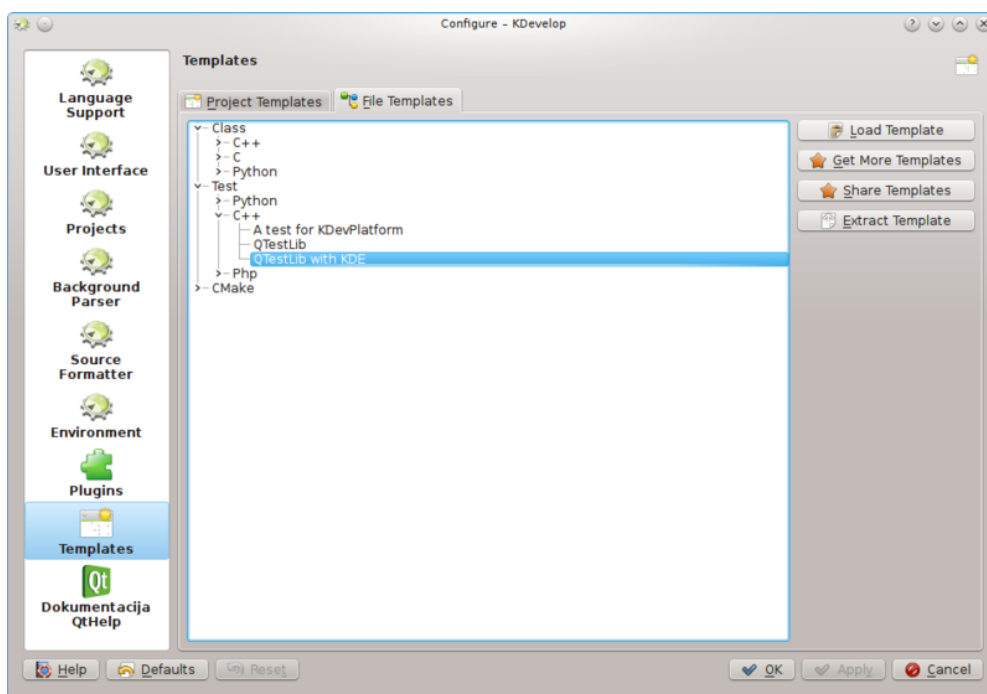
4.3 Інші файли

Хоча роботу засобів створення коду на основі шаблонів акцентовано на створенні класів та тестів модулів, подібну методику можна застосувати для створення будь-яких фрагментів файлів коду. Наприклад, можна скористатися шаблонами для модулів пошуку CMake або файлів `.desktop`. Для цього просто виберіть пункт **Створити з шаблону** [#8230](#), а потім позначте відповідну категорію і шаблон. Якщо шаблон не належить до категорій **Клас** або **Перевірка**, вам достатньо буде вибрати умови ліцензування, значення додаткових параметрів

шаблону та розташування файлів результату. Як і для класів з тестами, на завершення роботи допоміжної програми буде створено відповідні файли і відкрито їх у вікні редактора.

4.4 Керування шаблонами

Ви можете отримати додаткові файли шаблонів безпосередньо з вікна, яке можна відкрити за допомогою пункту меню **Файл** → **Створити з шаблону**; достатньо натиснути кнопку **Отримати додаткові шаблони...** У відповідь буде відкрито вікно отримання нових даних, за допомогою якого ви зможете встановити додаткові шаблони, оновити вже встановлені шаблони або вилучити непотрібні вам шаблони. Також передбачено модуль налаштування для шаблонів. Доступ до цього модуля можна отримати за допомогою пункту меню **Параметри** → **Налаштувати KDevelop** → **Шаблони**. За допомогою цього модуля ви можете керувати шаблонами файлів (описано вище) і шаблонами проєктів (використовується для створення проєктів).



Звичайно ж, якщо потребам вашого проєкту не відповідає жоден шаблон, ви можете створити новий. Ймовірно, найпростішим способом буде копіювання вже створеного шаблону з наступним внесенням до нього змін. Вам можуть допомогти короткі **настанови** та розгорнутий **довідковий документ**. Щоб скопіювати встановлений шаблон, відкрийте вікно керування шаблонами за допомогою пункту меню **Параметри** → **Налаштувати KDevelop...** → **Шаблони**, виберіть шаблон, який слід скопіювати, потім натисніть кнопку **Видобути шаблон**. Виберіть теку призначення, потім натисніть кнопку **Гаразд**. Вміст шаблону буде видобуде до вказаної вами теки. Тепер ви можете почати редагування шаблону: відкривати і змінювати видобуті файли. Щойно потрібні зміни буде внесено, ви можете імпортувати ваш новий шаблон до KDevelop: відкрийте засіб керування шаблонами, перейдіть на відповідну вкладку (**Шаблони проєктів** або **Шаблони файлі**) і натисніть кнопку **Завантажити шаблон**. Відкрийте файл опису шаблону, тобто файл з суфіксом назви `.kdevtemplate` або `.desktop`. KDevelop стисне всі файли до архіву шаблону і імпортує його дані.

ПРИМІТКА

Якщо ви копіюєте вже створений шаблон, не забудьте перейменувати його до повторного імпортування. Якщо ви цього не зробите, попередній шаблон буде перезаписано або у списку з'являться два шаблони з однаковими назвами. Щоб перейменувати шаблон, перейменуйте файл опису так, щоб він мав унікальну назву (суфікс слід зберегти), і змініть запис Name у файлі опису.

Якщо вам потрібно створити шаблон «з нуля», ви можете почати зі зразка шаблону класу C++: [створіть новий проєкт](#) і виберіть проєкт Шаблон класу C++ у категорії KDevelop.

Розділ 5

Збирання (компіляція) проєктів з нетиповими Makefile

У багатьох проєктах спосіб збирання файлів коду та визначення тих файлів, які слід повторно зібрати у разі внесення змін до коду, виконується за допомогою файлів Makefile, обробку яких здійснює програма **make** (див., наприклад, [GNU make](#)). У простих проєктах нескладно створити такий файл власноруч. У великих проєктах створення таких файлів часто покладається на **GNU autotools** (autocconf, autoheader, automake). У цьому розділі ми припустимо, що файл Makefile вашого проєкту вже створено, вам просто потрібно вказати KDevelop, у який спосіб слід взаємодіяти з цим файлом.

ПРИМІТКА

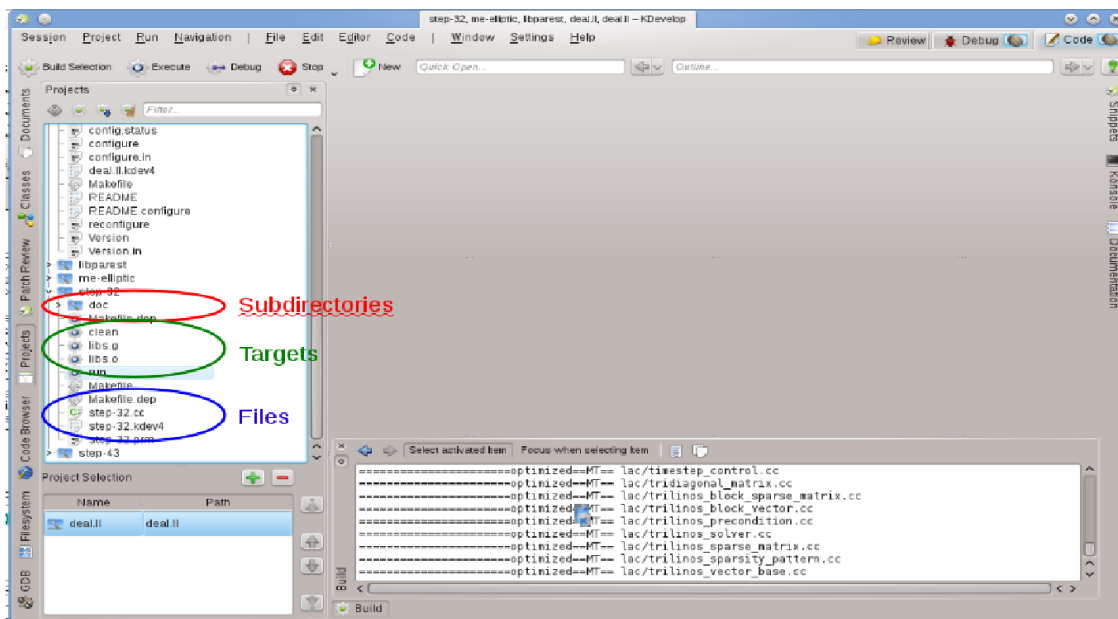
KDevelop 4.x не може працювати з **GNU autotools** безпосередньо на час написання цього підручника. Якщо відповідні інструменти використовуються у вашому проєкті, вам доведеться запускати скрипт `./configure` або подібний до нього скрипт з командного рядка вручну. Якщо ви бажаєте зробити це з самого вікна KDevelop, відкрийте панель інструмента **Konsole** (якщо потрібно, додайте цю панель у нижній частині головного вікна за допомогою пункту меню **Вікна** → **Додати панель інструмента**), у якій можна буде віддати команду оболонці і вкажіть команду `./configure`.

Насамперед, слід повідомити KDevelop про цілі збирання у вашому файлі Makefile. Передбачено два способи: вибір окремих цілей Makefile і вибір набору цілей, які потрібно збирати доволі часто. Для реалізації обох цих способів слід відкрити вікно інструмента **Проєкти** натисканням заголовка вкладки **Проєкти**, розташованої у лівій частині головного вікна KDevelop (якщо цього заголовка немає у вікні вашої програми, вище наведено настанови щодо додавання кнопки відповідного інструмента). Панель **Проєкти** складається з двох частин. У верхній частині з заголовком **Проєкти** показано список всіх ваших проєктів, у якому можна розгорнути пункти каталогів. У нижній частині з заголовком **Вибір проєкту** показано список набори проєктів, які буде зібрано, якщо ви виберете пункт меню **Проєкт** → **Зібрати позначене** або натиснете клавішу **F8**; нижче ми поговоримо про цю частину докладніше.

5.1 Збирання окремих цілей з Makefile

У верхній частині панелі проєктів розгорніть список одного з проєктів, наприклад, того, для якого слід виконати збирання однієї з цілей Makefile. У списку ви побачите піктограми каталогів, файлів у каталозі верхнього рівня проєкту, цілі Makefile, які вдалося визначити KDevelop. На знімку вікна ці категорії показано праворуч. Зауважте, що KDevelop певною

мірю *розуміє* синтаксис Makefile і тому сам визначає цілі, визначені у файлі Makefile (хоча це розуміння і має певні обмеження: не буде показано складені або неявні цілі).





Щоб зібрати будь-яку з цілей у списку, наведіть на її пункт вказівник миші, кладіть правою кнопкою миші і виберіть у контекстному меню пункт **Зібрати**. Наприклад, виконання цих дій для цілі «clean» призведе до виконання команди «make clean». Повідомлення щодо виконання дій можна буде побачити у підвікні **Збирання**. Це вікно відповідає інструменту **Зібрати**, отже його можна закрити і пізніше відкрити за допомогою пункту **Зібрати**, розташованого на панелі інструментів головного вікна. На нашому знімку цю кнопку розташовано праворуч внизу.

5.2 Вибір збірки цілей з Makefile для регулярного збирання

Клацання правою кнопкою миші на окремих пунктах цілей швидко втомлює. Набагато простіше було б створити для одного або декількох проектів окремі цілі, які постійно збираються протягом розробки. З цією метою у середовищі реалізовано «збирання позначених цілей»: збирання наборів цілей Makefile у певному порядку у відповідь на натискання кнопки **Зібрати позначене** у верхній частині вікна, вибір пункту меню **Проект** → **Зібрати позначене** або натискання клавіші **F8**.

Список позначених цілей Makefile показано у нижній частині панелі **Проекти**.

Типово, позначено буде всі проекти, але ви можете змінити список. Наприклад, якщо у вашому списку проектів три проекти (базова бібліотека L і дві програми, A і B), але зараз ви працюєте лише над проектом A, ви можете вилучити проект B зі списку: позначте його пункт і натисніть кнопку . Крім того, вам, ймовірно, захочеться, щоб бібліотеку L було зібрано перед проектом A. Змініть порядок пунктів у списку за допомогою кнопок, розташованих праворуч від списку. Ви також можете додати певну ціль Makefile до позначених: кладіть правою кнопкою миші у вільному місці списку і виберіть у контекстному меню пункт **Додати до набору збирання** або просто позначте відповідний пункт і натисніть кнопку , розташовану над списком позначених цілей.

У KDevelop передбачено можливість визначення дій під час збирання позначених цілей. Щоб виконати налаштування, скористайтеся пунктом **Проект** → **Відкрити налаштування**. У налаштуваннях ви, наприклад, можете визначити кількість завдань, які «make» може виконувати одночасно (якщо на вашому комп'ютері встановлено, скажімо, 8 процесорних ядер,

введення числа 8 у відповідне поле буде доречним вибором). У діалоговому вікні **Типовою** ціллю **make** є ціль Makefile, використана для *всіх* цілей у позначеному наборі.

5.3 Обробка повідомлень про помилки

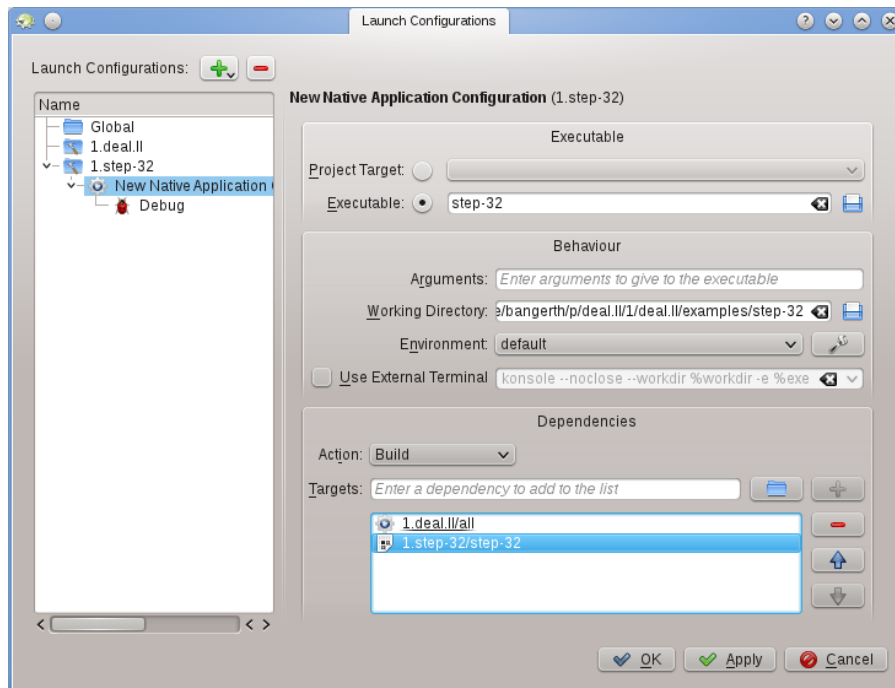
Якщо під час компіляції буде виявлено помилку, просто наведіть вказівник миші на повідомлення про помилку і клацніть лівою кнопкою миші. Курсор буде переведено до рядка (і, якщо вказано, позиції у рядку), де було виявлено помилку. Залежно від типу помилки, KDevelop може також запропонувати вам декілька варіантів дій для її виправлення, наприклад, за допомогою оголошення неоголошеної змінної, якщо таку змінну було виявлено.

Розділ 6


Запуск програм у KDevelop

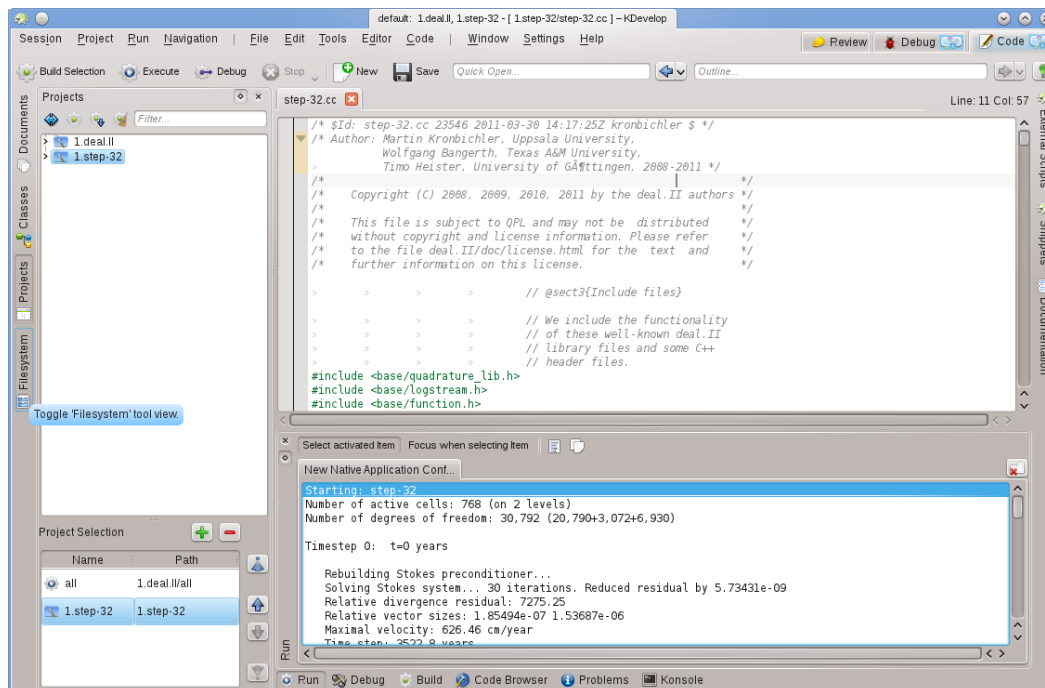
Після збирання програми її слід запустити. Щоб зробити це, вам слід налаштувати *інструменти запуску* для ваших проєктів. *Інструмент запуску* складається з назви виконуваного файлу, набору параметрів командного рядка та середовища виконання (наприклад, «запустити програму у командній оболонці» або «запустити цю програму за допомогою інструменту для усунування вад»).

6.1 Налаштування запуску у KDevelop



Щоб налаштувати інструменти запуску, скористайтесь пунктом меню **Виконання** → **Налаштувати запуски**, позначте у списку проєкт, до якого ви хочете додати інструмент запуску і натисніть **+**. Потім введіть назву виконуваного файлу і адресу каталогу, з якого має бути запущено програму. Якщо запуск виконуваного файлу залежить від результату збирання виконуваного файлу і/або інших бібліотек, ви можете додати їх назви до списку,

розташованого у нижній частині вікна: виберіть пункт **Зібрати** зі спадного меню, потім натисніть кнопку  праворуч від поля для введення тексту і виберіть ціль для збирання. У нашому прикладі вибрано ціль **все** з проєкту *1.deal.II* і *step-32* з проєкту *1.step-32* з метою забезпечити одночасне збирання основної бібліотеки та програми на її основі до запуску програми. За допомогою цього ж діалогового вікна можна налаштувати налагоджування під час запуску: натисніть пункт **Налагоджування** і додайте назву виконуваного файлу зневадника. Якщо цим зневадником є типовий зневадник системи (наприклад, gdb у Linux®), вам не доведеться нічого додатково визначати.



Тепер можна спробувати запустити програму: скористайтесь пунктом меню головного вікна KDevelop **Виконання** → **Виконати запуск** (або натисніть комбінацію клавіш **Shift-F9**) і вашу програму буде запущено у окремому підвікні KDevelop. На наведеному вище знімку показано результат: нове підвікно **Виконання**, у якому можна бачити результати роботи запущеної програми, у нашому прикладі *step-32*.

ПРИМІТКА

Якщо вами було налаштовано декілька інструментів запуску, ви можете вибрати той з них, який запускатиметься у відповідь на натискання комбінації клавіш **Shift-F9**: скористайтесь пунктом меню **Виконання** → **Поточні налаштування запуску**. Існує неочевидний спосіб зміни назви налаштувань: у діалоговому вікні, яке можна відкрити за допомогою пункту меню **Виконання** → **Поточні налаштування запуску** двічі клацніть лівою кнопкою миші на пункті назви налаштувань у ієрархії ліворуч. Після цього ви зможете змінити назву налаштувань.

6.2 Деякі корисні клавіатурні скорочення

Запуск програми	
F8	Зібрати (викликати make)
Shift-F9	Виконати

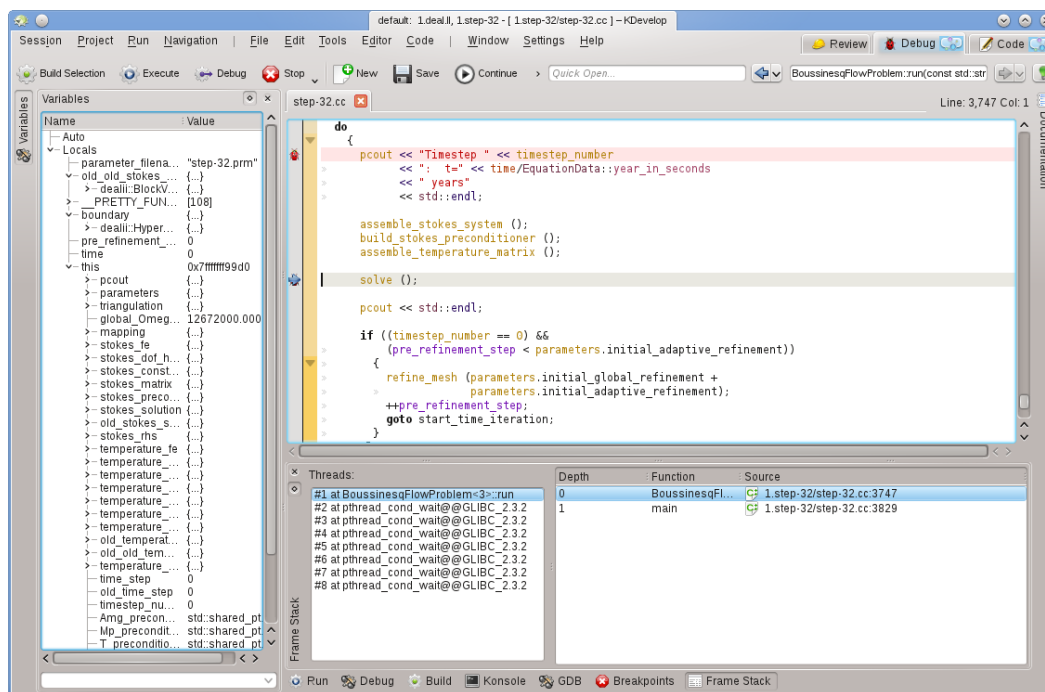
Alt-F9	Виконати програму під керування зневадника. Перед таким запуском варто встановити точки зупинки: для цього достатньо клацнути правою кнопкою у відповідному рядку коду і вибрати пункт контекстного меню, пов'язаний зі встановленням точки зупинки.
---------------	--

Розділ 7

Налагоджування програм у KDevelop

7.1 Запуск програми під керуванням програми для налагоджування

Після того, як запуск програми буде налаштовано (див. [Запуск програм](#)), ви зможете запускати програму під керуванням інструмента налагоджування: скористайтеся пунктом меню **Виконання** → **Налагоджувальний запуск** або натисніть комбінацію клавіш **Alt-F9**. Якщо ви знайомі з роботою gdb, результат буде той самий, що і після запуску gdb з вказаним виконуваним файлом у налаштуваннях запуску з наступною командою **Run**. Це означає, що якщо програмою буде десь викликано `abort()` (наприклад, якщо оператором контролю буде виявлено помилку) або якщо буде виявлено помилку сегментування, інструмент налагоджування зупинить роботу програми. З іншого боку, якщо програма зможе виконати роботу до кінця (правильно чи неправильно), інструмент налагоджування не зупинить її роботу, аж доки ця робота не завершиться сама. У такому разі вам можуть знадобитися встановлені ще до запуску програми точки зупинки у всіх рядках коду, де інструмент налагоджування має зупинити роботу програми. Встановити такі точки зупинки можна встановленням курсора у відповідному рядку з наступним вибором пункту меню **Виконання** → **Встановити/зняти точку зупинки** або використанням пункту контекстного меню (викликається клацанням правою кнопкою миші) **Встановити/зняти точку зупинки**.

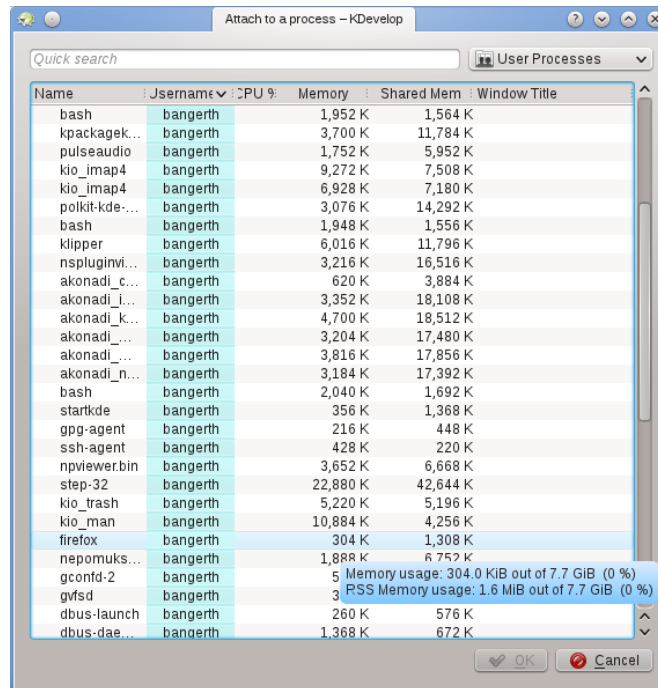


Запуск програми у зневаднику переведе KDevelop у інший режим: всі кнопки інструментів на периметрі головного вікна програми буде замінено на відповідні кнопки налагоджування, а не редагування. Визначити режим, у якому працює програма можна за верхньою правою частиною вікна: там ви побачите вкладки з назвами **Перегляд**, **Налагоджування** та **Код**. Натискання заголовків цих вкладок надає змогу перемикатися між трьома режимами. У кожного з режимів є власний набір інструментів, налаштувати їхній перелік можна у такий само спосіб, у який ми налаштовували інструменти режиму **Код** у розділі **Інструменти та панель перегляду**.

Після зупинки інструмента налагоджування (у точці зупинки або у точці виклику `abort()`) ви зможете вивчити різноманітні дані щодо роботи програми. Наприклад, на наведеному вище зображенні нами вибрано інструмент **Стек викликів** у нижній частині (приблизний еквівалент команд «backtrace» та «info threads» у gdb), отже показано список запущених потоків виконання ліворуч (у нашому прикладі таких потоків 8) та спосіб переходу до поточної точки зупинки праворуч (у нашому прикладі `main()` викликано `run()`; список був би довшим, якби ми зупинилися у функції, викликаній з `run()`). Ліворуч можна бачити локальні змінні, зокрема поточний об'єкт (об'єкт, на який вказує змінна `this`).

Після зупинки ви можете діяти у декілька способів: наказати виконати поточний рядок (**F10**, команда «next» gdb), увійти до функції (**F11**, команда «step» у gdb) або виконати інструкції до кінця функції (**F12**, команда «finish» gdb). На кожному з етапів виконання KDevelop оновлюватиме значення змінних, показані ліворуч. Ви також можете просто навести вказівник миші на частину коду, наприклад, назву змінної, і KDevelop покаже поточне значення та запропонує зупинити виконання програми під час наступної зміни значення змінної. Якщо ви знайомі з gdb, ви також можете натиснути кнопку інструмента **GDB** у нижній частині вікна і отримати змогу вводити команди gdb безпосередньо, наприклад, для того, щоб змінити значення змінної (у поточній версії середовища, здається, не передбачено іншого способу).

7.2 Приєднання програми для налагоджування до запущеного процесу



Іноді виникає потреба у налагодженні програми, яку вже запущено. Одним з випадків такої потреби є налагодження паралельно запущених за допомогою **MPI** програм або налагодження довготривалого фонових процесу. Виконати таке налагодження можна за допомогою пункту меню **Виконання** → **Долучити до процесу**, у відповідь на вибір якого буде відкрито вікно, подібне до наведеного вище. Вам потрібно вибрати програму, яка відповідає поточному відкритому проекту у KDevelop — у нашому випадку такою програмою буде `step-32`.

Цей список програм може бути доволі довгим, як у нашому прикладі. Ви можете дещо спростити собі роботу за допомогою спадного списку у верхній частині вікна. Типовим пунктом цього списку є **Процеси користувачів**, тобто всі програми, запущені всіма користувачами, які працюють у системі (якщо ви працюєте за власним стаціонарним або портативним комп'ютером, ймовірно, ви є єдиним користувачем, окрім користувача `root` та різноманітних облікових записів служб); у списку не буде процесів користувача `root`. Обмежити перелік процесів можна за допомогою пункту **Власні процеси**, його вибір призведе до вилучення зі списку програм, запущених іншими користувачами. Кращим варіантом є вибір пункту **Лише програми**, — зі списку буде вилучено багато процесів, які формально запущено від вашого імені, але які ви власноруч не запускали, зокрема програма для керування вікнами, фонові завдання тощо навряд чи є достойними кандидатами для налагодження.

Після вибору процесу, долучення до нього переведе KDevelop у режим налагодження, відкриє всі звичайні панелі налагодження і зупинить програму на рядку, який виконувався під час долучення. На цьому етапі доцільно встановити точки зупинки, точки перегляду або налаштувати інші елементи стеження за виконанням, а потім продовжити виконання програми за допомогою пункту меню **Виконання** → **Продовжити**.

7.3 Деякі корисні клавіатурні скорочення

Налагоджування	
F10	Перейти до наступної інструкції ("next" у gdb)
F11	Увійти у наступну інструкцію ("step" у gdb)
F12	Вийти з інструкції ("finish" у gdb)

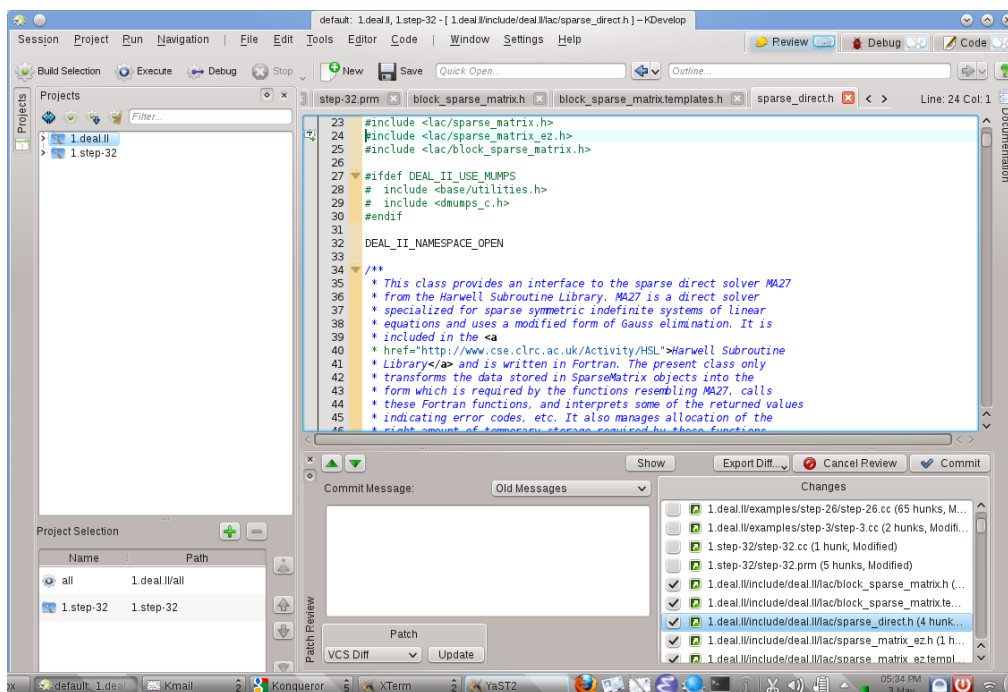
Розділ 8

Робота з системами керування версіями

Якщо ви працюєте з доволі великими проектами, ймовірно, код проекту керується системою керування версіями, наприклад [subversion](#) або [git](#). Наведені нижче настанови відповідають **subversion**, але їх може бути використано для **git** або будь-якої іншої підтримуваної системи керування версіями.

По-перше, якщо вміст каталогу вашого проекту є наслідком використання певної системи керування версіями, KDevelop автоматично визначить це. Іншими словами, не потрібно вказувати KDevelop, що середовищу слід отримати код з системи керування версіями самостійно, — достатньо вказати KDevelop каталог, у якому зберігається вже отримана копія сховища. Якщо у вас є такий каталог, відкрийте панель **Проекти**. За її допомогою ви зможете виконати декілька корисних дій:

- Якщо вміст вашого каталогу застарів, ви можете оновити дані зі сховища: клацніть на пункті назви проекту правою кнопкою миші, відкрийте у контекстному меню підменю **Subversion** і скористайтеся пунктом **Оновити**. Таким чином, ви зможете підтримувати вміст проекту у найактуальнішому стані, відповідно до вмісту сховища коду.
- Якщо вам потрібно виконати оновлення для якогось окремого підкаталогу або певних файлів, розгорніть список файлів проекту, знайдіть у ньому потрібні пункти і виконайте для тих описану вище дію.



- Якщо ви внесли зміни до одного або декількох файлів, розкрийте на панелі вмісту проекту каталог з цими файлами і клацніть на пункті каталогу правою кнопкою миші. У відповідь буде відкрито меню **Subversion** з декількома пунктами варіантів дій. Виберіть пункт **Порівняти з Базовим**, щоб переглянути відмінності між редагованою вами версією і версією, яка зберігається у сховищі (версією «base»). На панелі перегляду, яку буде відкрито, ви зможете переглянути відмінності («diff») для всіх файлів у каталозі.
- Якщо зміни було внесено лише до одного файла, ви можете відкрити меню **Subversion** для цього файла простим клацанням правою кнопкою миші на пункті файла на панелі перегляду проекту. Можна зробити ще простіше: клацніть правою кнопкою миші на панелі редактора, на якій відкрито файл, — у контекстному меню ви побачите відповідний пункт.
- Якщо вам потрібно надіслати до сховища один або декілька змінених файли, клацніть правою кнопкою миші на пункті кожного з файлів, підкаталогів або всього проекту і виберіть у контекстному меню пункт **Subversion** → **Надіслати**. Після вибору цього пункту середовище буде переведено у режим **Перегляд**, третій режим, пункт якого, поряд з пунктами **Код** і **Налагоджування** можна бачити у правому верхньому куті головного вікна KDevelop. На наведеній ілюстрації показано зразок вікна у такому режимі. У режимі **Перегляд** у верхній частині вікна буде показано відмінності у файлах всього підкаталогу або проекту, пункт кожного зміненого файла буде позначено кольором (див. різні вкладки у цій частині вікна). Типово, всі змінні файли буде додано до набору змін, який буде надіслано до сховища, але ви можете зняти позначення з частини файлів, якщо внесені до них зміни не пов'язано з окремою порцією змін, які ви надсилаєте. Наприклад, у нас знято позначення з пунктів `step-32.cc` і `step-32.prm`, оскільки зміни у цих файлах не пов'язано з іншими змінами у проекті, отже цього разу їх не буде надіслано до сховища (їх можна буде надіслати пізніше окремим внеском). Після перегляду змін ви можете вказати повідомлення щодо внеску у полі для введення тексту і натиснути кнопку **Надіслати**, розташовану праворуч, щоб надіслати зміни до сховища.
- Під час перегляду відмінностей, якщо потрібно надіслати зміни лише до одного файла, ви можете просто клацнути правою кнопкою миші на відповідному пункті панелі редактора і вибрати у контекстному меню пункт **Subversion** → **Надіслати**.

Розділ 9

Налаштовування KDevelop

Іноді корисно змінити типовий вигляд або спосіб роботи KDevelop, наприклад, ви звикли до інших клавіатурних скорочень або у вашому проєкті використовується інші відступи у коді. У наступних розділах ми коротко обговоримо різні способи налаштування KDevelop відповідно до ваших потреб.

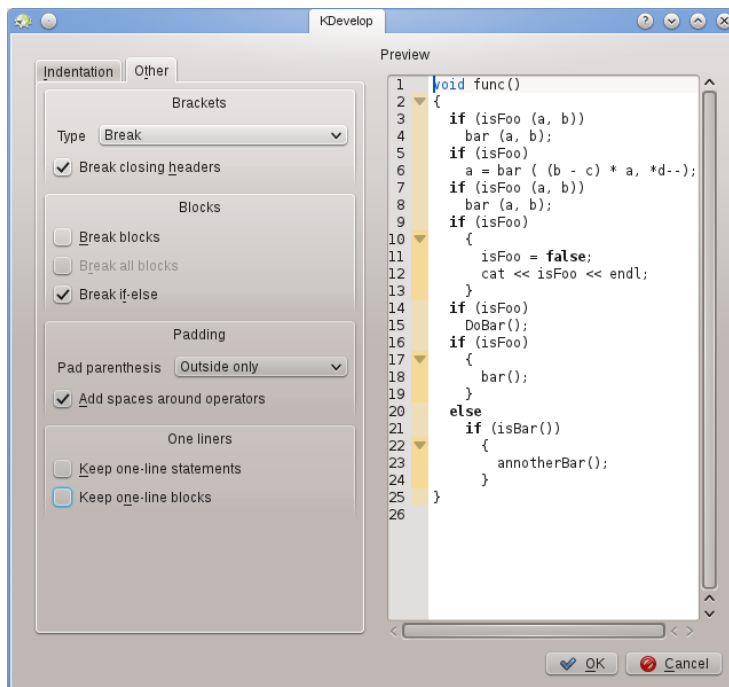
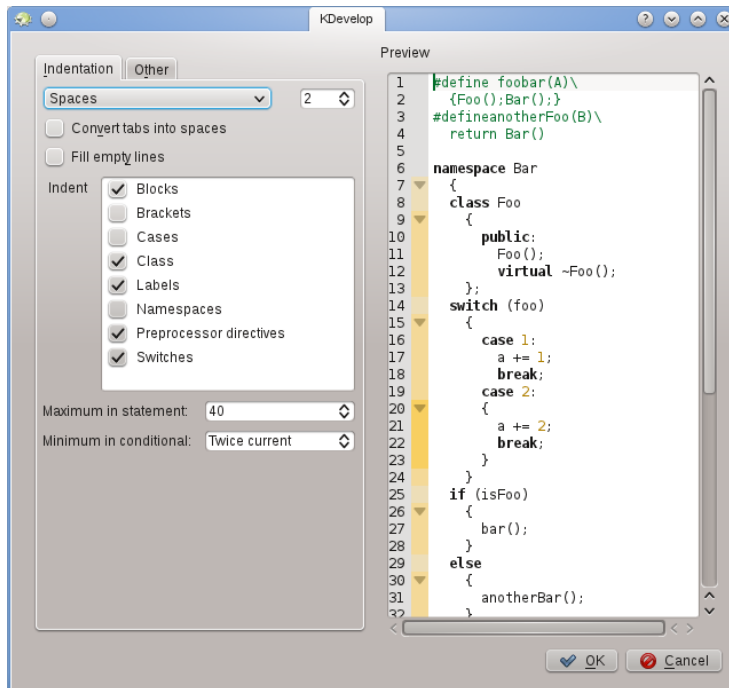
9.1 Налаштовування редактора

Корисно налаштувати вбудований редактор та інші компоненти KDevelop. Однією з загальних речей є вмикання показу нумерації рядків за допомогою пункту меню **Редактор** → **Перегляд** → **Показати номери рядків**. У такому режимі простіше буде визначати рядки з помилками під час компіляції або визначати відповідність діагностичних повідомлень позиціям у коді. У тому ж підменю ви можете увімкнути «Рамку для піктограм» — стовпчик у лівій частині панелі редактора, у якому KDevelop показуватиме піктограми, зокрема позначок точок зупинки у рядках.

9.2 Налаштовування відступів у коді

У кожного з нас є певні уподобання щодо форматування коду. У багатьох проєктах є власні правила щодо відступів у коді. Ці правила можуть не відповідати правилам встановлення відступів KDevelop. Ви можете просто змінити правила відступів: скористайтеся пунктом меню **Параметри** → **Налаштувати KDevelop**. Перейдіть на сторінку **Форматування коду** за допомогою списку ліворуч. Ви можете вибрати один з попередньо визначених розповсюджених типів форматування або визначити власний стиль додаванням нового стилю з наступним його редагуванням. Можливо, повністю відтворити стиль форматування коду вашого проєкту у минулому не вдасться, але ви можете гранично до нього наблизитися за допомогою параметрів нового стилю. Приклад наведено на двох зображеннях нижче.

Підручник з KDevelop



ПРИМІТКА

У KDevelop 4.2.2 ви можете створити стиль для файлів окремого типу MIME (наприклад, для стилю файлів заголовків C++), але цей стиль не буде показано у списку можливих стилів для файлів інших типів MIME (наприклад, для коду файлів C++), хоча, звичайно ж, було б корисно використати цей тип для обох типів файлів. Тому вам доведеться визначити стиль двічі: один раз для файлів заголовків і один раз для файлів коду. Про цю ваду було повідомлено розробникам, [KDevelop, вада 272335](#).

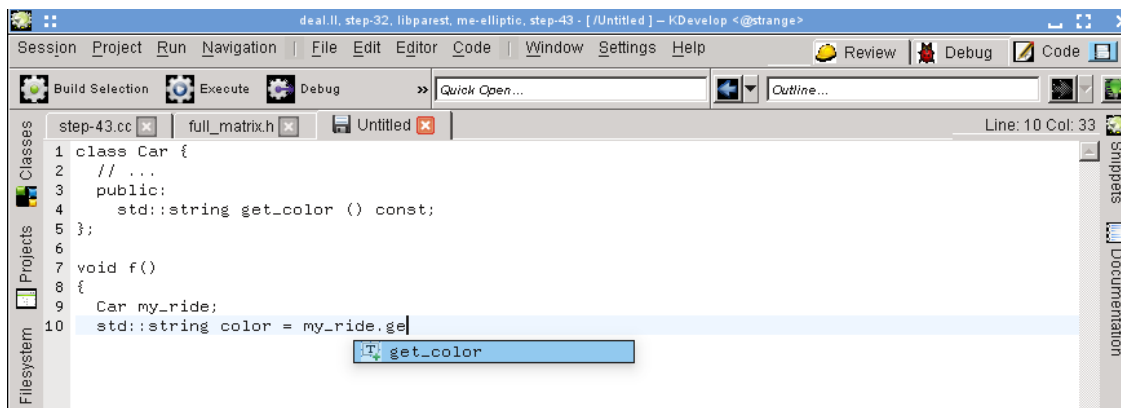
9.3 Налаштовування клавіатурних скорочень

У KDevelop передбачено майже безмежний список клавіатурних скорочень (деякі з них можна знайти у списках «Корисні клавіатурні скорочення» у декількох частинах цього підручника). Ці клавіатурні скорочення можна змінити за вашими потребами за допомогою пункту меню **Параметри** → **Налаштувати скорочення**. У верхній частині діалогового вікна ви можете вказати ключ пошуку, щоб переглянути лише відповідні ключу скорочення. Після цього ви можете змінити клавіатурні скорочення для бажаних команд.

Однією з двох корисних змін буде встановлення для **Вирівняти** скорочення **Tab** (багато хто звичайно не додає табуляції вручну і віддає перевагу вибору форматування коду редактором; якщо скорочення буде змінено, після натискання **Tab** KDevelop виконає додавання відступів, вилучення відступів або вирівнювання коду). Другою зміною буде встановлення для команди **Встановити/зняти точку зупинки** скорочення **Ctrl-B**, оскільки встановлення або зняття точки зупинки є доволі поширеною дією.

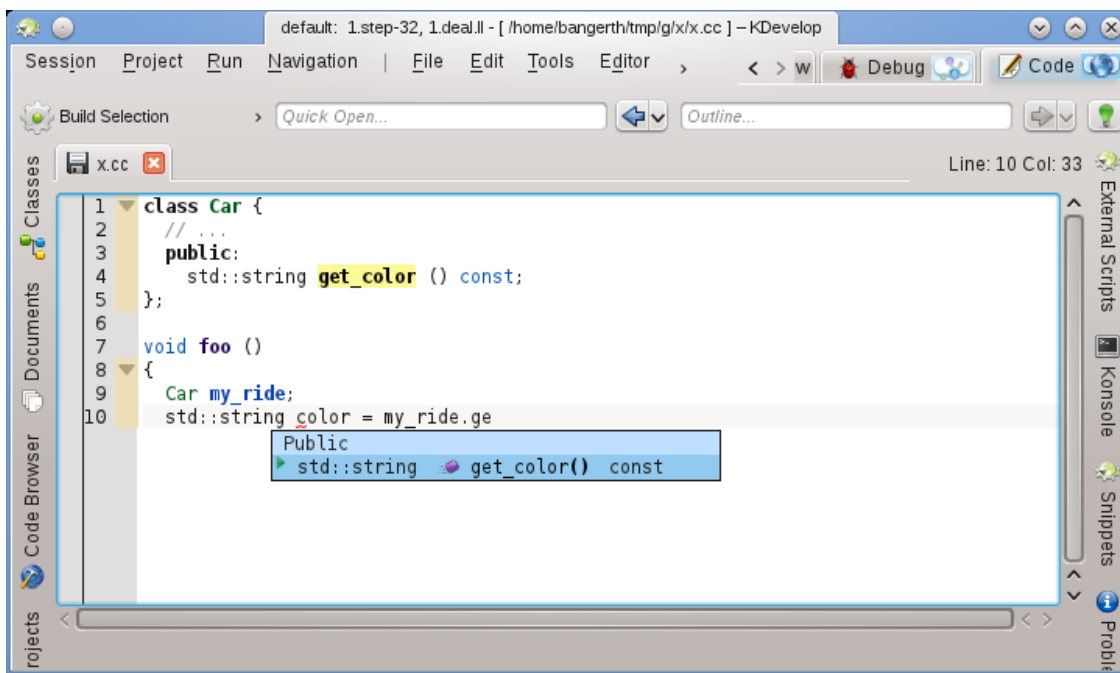
9.4 Налаштування автодоповнення коду

Можливості з автоматичного доповнення коду було обговорено у [цьому розділі підручника, присвяченому створенню коду](#). У KDevelop автоматичне доповнення відбувається з двох джерел: редактора і рушія обробки. Редактор (Kate) є компонентом більшого середовища KDE, він пропонує доповнення на основі інших частин того самого документа. Таке автоматичне доповнення можна впізнати на панелі підказки за піктограмою, вказаною перед відповідним варіантом:



Автоматичне доповнення коду редактором можна налаштувати за допомогою пункту меню **Параметри** → **Налаштувати редактор** → **Редагування** → **Автозавершення**. Зокрема, ви можете визначити кількість символів, які має бути введено у слові для вмикання механізмів автоматичного доповнення.

З іншого боку, автоматичне доповнення самої програми KDevelop є набагато потужнішим, оскільки у ньому використано семантичні дані щодо контексту слова. Наприклад, середовище пропонує лише ті функції для коду `object.`, які використано у відповідному об'єкті тощо. Ось приклад:



Дані щодо контексту надходять з декількох додатків підтримки мов програмування, якими можна скористатися після того, як файл буде збережено (середовище визначить мову програмування за типом збереженого файла).

Автоматичне доповнення KDevelop працюватиме під час введення вами коду майже всюди, де таке автоматичне доповнення можливе. Налаштувати механізм доповнення можна за допомогою пункту меню **Параметри** → **Налаштувати KDevelop** → **Підтримка мов**. Якщо пункт **Увімкнути автоматичний виклик** ще не позначено (його має бути типово позначено), позначте його.

У KDevelop передбачено два способи показу доповнення: у режимі **Мінімальне автоматичне доповнення** середовище показує на панелі підказки лише основні дані (тобто простір назв, клас, функцію або назву змінної). Таке доповнення подібне до доповнення у Kate (окрім піктограми).

У режимі **Повне доповнення** середовище додатково показує тип кожного запису та, у разі якщо доповнюється функція, перелік аргументів. Крім того, якщо ви заповнюєте аргументи функції, у режимі повного доповнення буде показано додаткову інформаційну панель над курсором, де буде наведено дані щодо поточного аргументу.

У режимі автоматичного доповнення коду у KDevelop також має бути наведено у верхній частині списку та підсвічено зеленим всі пункти доповнення, які відповідають поточному очікуваному типу (найкращі відповідники) у режимах мінімального та повного доповнення.

У діалоговому вікні налаштування передбачено три варіанти рівня автоматичного доповнення:

- **Завжди мінімальне доповнення:** ніколи не показувати «повного доповнення».
- **Мінімальне автоматичне доповнення:** показувати «повне доповнення», якщо автоматичне доповнення було викликано вручну (тобто було натиснуто **Ctrl-Пробіл**)
- **Завжди повне доповнення:** завжди показувати «повне доповнення».

Розділ 10

Подяки і ліцензія

Список власників авторських прав на документацію до програми можна знайти у [журналі сторінки KDevelop4/Manual UserBase](#).

Переклад українською: Юрій Черноіван yurchor@ukr.net

Цей документ поширюється за умов дотримання [GNU Free Documentation License](#).