Міністерство освіти і науки України
Чорноморський державний університет імені Петра Могили

С. В. Пузирьов

ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ МЕРЕЖ ІЗ ВИКОРИСТАННЯМ БІБЛІОТЕКИ *wxWidgets*

Методичні рекомендації
до виконання практичних та самостійних робіт з дисципліни
«ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ МЕРЕЖ»
для студентів спеціальності 7.05010202 – системне програмування всіх форм навчання

Випуск 240

Миколаїв – 2015

УДК 004.4’2
ББК 32.973-01
П 88

Рекомендовано вченою радою Чорноморського державного університету імені Петра Могили (протокол № 12 від 2 липня 2015 року).

Рецензенти:

Мусієнко М. П., д-р техн. наук, професор, декан факультету комп’ютерних наук ЧДУ ім. Петра Могили;

Журавська І. М., канд. техн. наук, доцент кафедри інформаційних технологій та програмних систем ЧДУ ім. Петра Могили.

П 88 **Пузирьов С. В.**

Програмування комп’ютерних мереж із використанням бібліотеки *wxWidgets* : [методичні рекомендації до виконання практичних та самостійних робіт з дисципліни «Програмування комп’ютерних мереж»] / С. В. Пузирьов. – Миколаїв : ЧДУ ім. Петра Могили, 2015. – 76 с.

Методичні рекомендації містять практичні та самостійні роботи, які охоплюють основи розробки мережевих програмних додатків із використанням вільного програм-ного забезпечення – середовища Code::Blocks та бібліотеки *wxWidgets*.

УДК 004.4’2
ББК 32.973-01

ISSN 1811-492X

ЗМІСТ

ВСТУП

Мережеве програмування є досить складною і водночас цікавою галуззю сучасної індустрії IT. Від надійності та ефективності мережевих додатків залежить сьогодні надійність роботи великих корпоративних та промислових систем. Тому знання основ мережевого програмування є необхідною складовою в підготовці фахівця із системного програмування.

Однією із суттєвих складностей мережевого програмування є залежність мережевих API від платформи операційної системи. Наприклад, *Windows* та *Linux* мають схожі програмні інтерфейси, однак вони містять незначні відмінності, які під час переходу на іншу платформу можуть створити значні труднощі в пошуку та усуненні помилок.

Тому, природно, постає необхідність використання в мережевому програмуванні платформонезалежних фреймворків, які дозволяють із мінімальними виправленнями переносити додатки з однієї платформи на іншу.

wxWidgets – це інструментарій для написання десктопних і мобільних застосувань із графічним інтерфейсом користувача (Graphical User Interface, GUI). Ця бібліотека є фреймворком, тобто вона реалізує велику частину роботи й поведінки вікон. Бібліотека *wxWidgets* містить величезну кількість класів і методів, які програміст може використати й налаштувати. Додаток зазвичай використовує вікно, що містить стандартні елементи управління, можливо, показує спеціальні зображення або графіку, а також реагує на введення з клавіатури, маніпулятора миші або з інших джерел. Додаток також може взаємодіяти з іншими процесами або управляти іншими програмами. Отже, *wxWidgets* дозволяє легко написати додатки, які можуть робити все те, що й сучасні програми.

Попри те, що *wxWidgets* часто називають «інструментарієм для розробки GUI», фактично бібліотека дозволяє зробити значно більше й може бути корисною в найрізноманітніших сферах розробки. Це є наслідком того, що додатки *wxWidgets* повинні мати можливість перенесення на різні платформи, а тому бібліотека містить класи для роботи з файлами й потоками введення/виведення, нитками, налаштуваннями додатка, міжпроцесорною комунікацією, файлами допомоги, базами даних і багатьма іншими.

Однією з особливостей, яка вигідно відрізняє *wxWidgets* від інших фреймворків, зокрема MFC або OWL, є її вражаюча переносимість. У *wxWidgets* є програмний інтерфейс для додатків (Application Programming Interface, API), який однаковий (чи майже однаковий) на всіх

підтримуваних платформах. Це означає, що додаток може бути написаний у системі *Windows* і з невеликими змінами (або навіть без них) скомпілюватися в *Linux* або *Mac OS X*. Ця обставина дає величезну перевагу, порівняно з повним переписуванням додатка під кожну платформу, у тому числі означає, що вам не треба вивчати різні API для кожної платформи. Крім того, бібліотека дозволяє йти в ногу з прогресом. Коли змінюється програмне середовище, *wxWidgets* змінюється разом із ним, дозволяючи додаткам переноситися на останні версії систем, підтримуючи велику частину їхніх нових можливостей.

Іншою приємною особливістю є те, що *wxWidgets* реалізує рідний вид і поведінку програм. Деякі фреймворки використовують один і той же код для елементів управління на різних платформах (можливо, використовуючи спеціальні теми, щоб елементи управління були схожі на рідні). На відміну від них, *wxWidgets* використовує рідні елементи скрізь, де це тільки можливо (і власний набір елементів інакше), тому елементи не просто схожі на рідні елементи платформи – вони ними і є. Це дуже важливо для кінцевого користувача, оскільки навіть найменша, майже непомітна різниця в поведінці програми, порівняно зі стандартом платформи, може створити відчуття деякої штучності програми.

Чом би не використати для цього Java? Попри те, що Java прекрасно підходить для web-додатків, вона не завжди є хорошим вибором для десктопу. Зокрема, додатки на C++, що використовують *wxWidgets*, швидші, мають природніший вигляд і передбачувану поведінку, а також простіші в установці та підтримці, оскільки не залежать від вередливої віртуальної машини Java. Використання C++ також дозволяє отримати доступ до низькорівневої функціональності, легше інтегрувати додаток з існуючим кодом на C і C++. Саме з цих причин дуже мало популярних застосовних програм базується на Java. Отже, *wxWidgets* дозволить вам писати швидкі додатки, які будуть зрозумілі користувачеві.

wxWidgets є проектом із відкритим початковим кодом. Зазвичай це означає, що не треба платити за його використання (доки ви не захочете зробити пожертвування проекту!), але також має дуже важливе філософське та стратегічне значення. Продукти з відкритим кодом завжди протиставляють їхнім пропріетарним еквівалентам. Використовуючи *wxWidgets*, ви завжди знаєте, що код, на який ви покладаєтеся, ніколи не зникне. Ви завжди зможете самостійно виправити будь-яку проблему, поправивши оригінальний код.

Бібліотека *wxWidgets* широко підтримується комп'ютерною індустрією. Список користувачів включає такі відомі компанії, як AOL, AMD, CALTECH, Lockheed Martin, NASA, Open Source Applications Foundation, Xerox і багато інших. *wxWidgets* об'єднує величезну кількість користувачів: від програмістів-одинаків до величезних корпорацій, від відділень комп'ютерних наук до центрів медичних досліджень і від екологічних організацій до індустрії телекомунікацій. Бібліотека використовується безліччю відкритих проектів, зокрема редактором музики Audacity й системою управління базами даних pgAdmin III.

wxWidgets використовують із різних причин: для одних – це проста й елегантна заміна MFC, для інших – можливість легко міняти платформу й перейти, наприклад, із *Microsoft Windows* на *Unix* або *Mac OS X*. Проект *wxWidgets* також приділяє увагу підтримці мобільних платформ. Існують порти для вбудовуваних систем на базі *Linux*, Microsoft Pocket PC і (можливо, порт скоро з'явиться) Palm OS.

Тема 1
Ознайомлення з бібліотекою *wxWidgets* та середовищем Code::Blocks

Теоретичні відомості

У таблиці 1.1 показно чотири концептуальні рівні: зовнішній API *wxWidgets*, основний порт, API платформи, використовуваний цим портом і, нарешті, назва операційної системи.

Таблиця 1.1

API портів *wxWidgets*

wxWidgets API								
Порт *wxWidgets*								
wxMSW	*wxGTK*	*wxX11*	*wxMotif*	*wxMac*	*wxCocoa*	*wxOS2*	*wxPalmOS*	*wxMGL*
API цільової платформи								
Win32	*GTK+*	*Xlib*	*Motif/ Lesstif*	*Carbon*	*Cocoa*	*PM*	*Palm OS, Protein APIs*	*MGL*
Операційна система								
Windows/ Windows CE	*Unix/Linux*			*Mac OS 9, Mac OS X*	*Mac OS X*	*OS/2*	*Palm OS*	*Unix/ DOS*

Нижче перераховано основні порти *wxWidgets*, які існували на час написання посібника.

- *wxMSW*

Цей порт компілюється та запускається на всіх 32- і 64-бітових версіях *Microsoft Windows*, включно з *Windows 95*, *Windows 98*, *Windows ME*, *Windows NT*, *Windows 2000*, *Windows XP* і *Windows 2003*. Він також може бути скомпільований під час використання бібліотеки *Winelib* під *Linux*. Також існує робоча конфігурація під *Windows CE*. Порт *wxMSW* може бути конфігурований для використання *wxUniversal* замість стандартних Win32 елементів.

- *wxGTK*

wxWidgets для *GTK+* може використати елементи *GTK+* версії 1.x або 2.x на будь-якому з різновидів *Unix*, де підтримується *X11* і *GTK+* (наприклад, *Linux*, *Solaris*, *HP-UX*, *IRIX*, *FreeBSD*, *OpenBSD*, *AIX* та інших). *wxGTK* є рекомендованим портом для всіх *Unix*-систем.

– *wxX11*

wxWidgets для *X11* використовує набір елементів із *wxUniversal* і запускається безпосередньо в *Xlib*, яка не має власних елементів управління. Ця обставина робить порт придатним для вбудовуваних систем, але при цьому можна використати і для побудови додатків, яким не хотілося б мати залежності від *GTK+*. Бібліотека дозволяє запускати додаток на будь-якій *Unix*-системі, де може бути запущений *X11*.

– *wxMotif*

Цей порт може використати *Motif, OpenMotif* або *Lesstif* на більшості *Unix*-систем.

– *wxMac*

wxMac спрямований на запуск додатків у середовищі *Mac OS 9* (потрібна версія не менша ніж 9.1) і *Mac OS X* (потрібна версія не менша ніж 10.2.8). Щоб компілювати додатки для *Mac OS 9*, знадобляться утиліти з *Metrowerks Code Warrior*. Для *Mac OS X* можна використати як *Metrowerks Code Warrior*, так і *Apple tools*. В останньому випадку необхідно використати *Xcode* версії 1.5 або вище. Якщо командний рядок вас не лякає, то можна обійтися компілятором *GCC* версії 3.3 або вище.

– *wxCocoa*

Цей порт ще знаходиться в процесі розробки. Призначений для використання *Cocoa API*, що вперше з'явилася в *Mac OS X*. Хоча функціональність *Carbon* і *Cocoa* дуже схожа, але цей порт має потенційну можливість підтримки середовища *GNUStep* на платформах, відмінних від *Mac*.

– *wxWinCE*

Порт *Windows CE* є оболонкою над різними SDK для систем на базі *Windows CE*, включно з *Pocket PC* і *Smartphone*. Основною частиною цього порту є порт для *Win32 wxMSW* з обмеженнями й доповненнями для роботи на малих платформах.

– *wxPalmOS*

Цей порт призначений для *Palm OS 6* (*Cobalt*). На сьогодні (на момент написання книги) він практично нефункціональний.

– *wxOS2*

Порт *wxOS2* є менеджером відображення для *OS/2* або *eComStation*.

– *wxMGL*

Цей порт спрямований на використання низькорівневого графічного програмного забезпечення *MGL* від *SciTech Software, Inc.* і використовує елементи *wxUniversal*.

Внутрішня організація

Усередині код *wxWidgets* розділений на шість шарів:

1. Загальний код використовується всіма портами. Він включає класи структур даних, інформацію про типи часу виконання й опис базових класів, таких як *wxWindowBase*, які використовуються для скорочення коду в усіх реалізаціях класів.

2. Похідний код реалізує розширені елементи управління незалежно від платформи, дозволяючи емулювати ці елементи та їхню функціональність, якщо вони не є наявними на цій платформі. *wxWizard* і *wxCalendarCtrl* є прикладом таких елементів.

3. *wxUniversal* – це множина базових елементів для платформ, у яких ці елементи відсутні, таких як *X11* і *MGL*.

4. Код, залежний від платформи, реалізує класи, які використовують існуючу функціональність. Наприклад, *wxMSW* реалізує *wxTextCtrl* як оболонку над звичайним елементом редагування для *Win32*.

5. Додатковий код знаходиться в окремому каталозі з ім'ям *contrib* і включає не обов'язкові, але корисні класи, такі як *wxStyledTextCtrl*.

6. Сторонній код складається з бібліотек, що розробляються незалежно від бібліотеки *wxWidgets*, але використовуються нею для реалізації важливих можливостей. Сторонній код включає бібліотеки *JPEG, Zlib, PNG* і *Expat*, а також деякі інші.

Кожен порт бере все, що йому необхідно з цих рівнів, і реалізує API бібліотеки *wxWidgets*.

Як *wxWidgets* дізнається, які класи необхідно використати під час компіляції? Коли ви включаєте в програму заголовні файли *wxWidgets*, такі як wx/textctrl.h, у програму включається визначуваний цільовою платформою файл (у випадку з *Windows* це файл wx/msw/textctrl.h) відповідно до директив у файлі wx/textctrl.h. Далі ваша програма лінкується разом із бібліотекою, яку було скомпільовано з відповідними налаштуваннями. Одночасно може існувати декілька конфігурацій бібліотеки, зокрема версії *Debug* і *Release*, а також динамічна й статична версії бібліотеки. У вас є можливість вимкнути деякі компоненти під час збирання *wxWidgets*, або зробити вибір між складками з підтримкою Unicode або ANSI, редагуючи для цього файл setup.h або використовуючи опції для конфігурації, вид яких залежить від використовуваного вами компілятора.

Важливим є те, що хоча *wxWidgets* і є оболонкою над кожним рідним API, але вона не забороняє писати специфічний для платформи код, якщо це необхідно. Проте така необхідність виникає дуже рідко.

Практична робота

1. Інсталяція програмного середовища *Code::Blocks*

Code::Blocks написаний на *C++* і використовує бібліотеку *wxWidgets*. Маючи відкриту архітектуру, може масштабуватися за рахунок підключуваних модулів. Підтримує мови програмування *C*, *C++*, *D* (з обмеженнями).

Інсталяційний пакет доступний за адресою:

http://www.codeblocks.org/downloads/.

Code::Blocks розробляється для *Windows*, *Linux* і *Mac OS X*. Середовище можна зібрати з вихідних кодів практично під будь-яку Unix-подібну систему, наприклад *FreeBSD*.

Інсталяційний пакет для *Windows* містить вільний *GNU* компілятор *MinGW*, який розміщується в папці *Code::Blocks*.

Нижче перераховано основні підтримувані *Code::Blocks* компілятори:

1. *MinGW / GCC C/C++(включений до інсталяції):*
   a. *GNU ARM GCC Compiler;*
   b. *GNU AVR GCC Compiler;*
   c. *GNU GCC Compiler for PowerPC;*
   d. *GNU GCC Compiler for TriCore.*
2. *Digital Mars C/C++.*
3. *Digital Mars D (із деякими обмеженнями).*
4. *SDCC (Small device C compiler).*
5. *Microsoft Visual C++ 6.*
6. *Microsoft Visual C++ Toolkit 2003.*
7. *Microsoft Visual C++ 2005/2008 (із деякими обмеженнями).*
8. *Microsoft Visual C++ 2010 (без підтримки відладчика, потрібно DDK).*
9. *Borland C++ 5.5.*
10. *Watcom.*
11. *Intel C++ compiler.*
12. *GNU Fortran.*
13. *GNU ARM.*
14. *GNU GDC.*

Усі ці компілятори потрібно підключати в *Code::Blocks* окремо, окрім *MinGW*.

Можливості інтерфейсу *Code::Blocks* такі:

- підсвічування синтаксису;
- згортання блоків коду;
- автодоповнення коду;
- браузер класів;

- планувальник під декілька користувачів;
- підтримка плагинів *devpack*;
- плагин *wxSmith* (інструмент швидкої розробки додатків (RAD) для *wxWidgets*);
- система перевірки правопису (тільки для коментарів);
- автоформатування коду *astyle code*.

Завантажте інсталяційний пакет *Code::Blocks* і встановіть його. Потім потрібно прописати змінну середовища *MINGW_HOME* у властивостях системи, яка містить шлях до папки з компілятором *MinGW*. Для цього натисніть *Ctrl+Break* та клікніть посилання *Дополнительные параметры системы*. У діалозі натисніть кнопку *Переменные среды…* (рис 1.1).

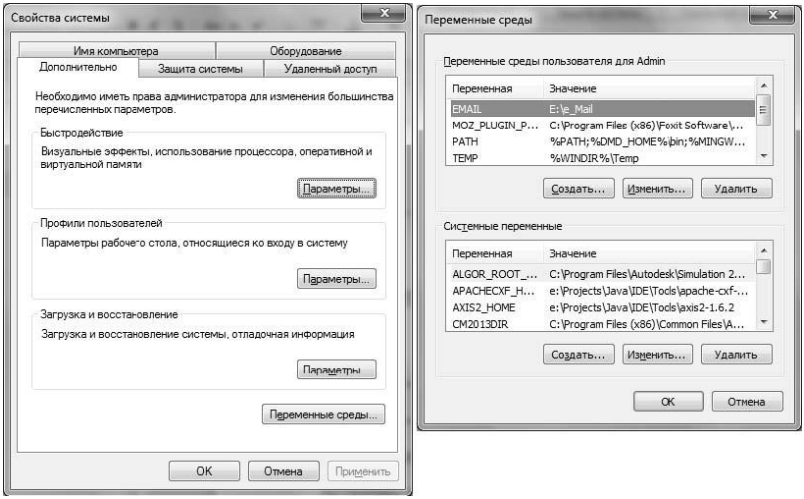

Рис. 1.1. Діалоги доступу до змінних середовища ОС

У новому діалозі натисніть *Создать…* і введіть ім'я змінної у верхнє поле, а шлях до компілятора – у нижнє (рис. 1.2).

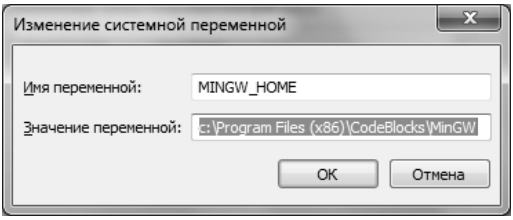

Рис. 1.2. Створення змінної *MINGW_HOME*

Далі натисніть кнопку *OK* і відредагуйте значення змінної *PATH* у верхній частині другого діалогу так (рис. 1.3):

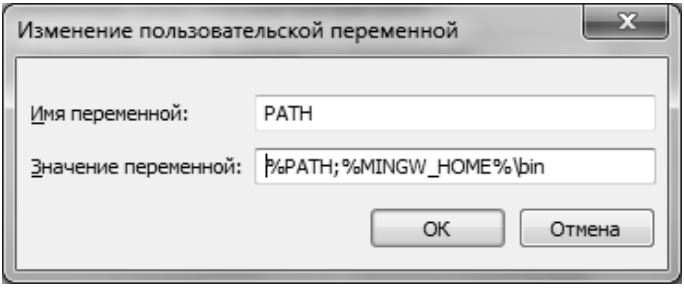

Рис. 1.3. Редагування змінної *PATH*

Після цього натисніть у всіх діалогах *OK*. Для перевірки правильності прописаних шляхів натисність *Win+R* та в діалозі запуску введіть *cmd* (рис. 1.4) і натисніть *OK*.

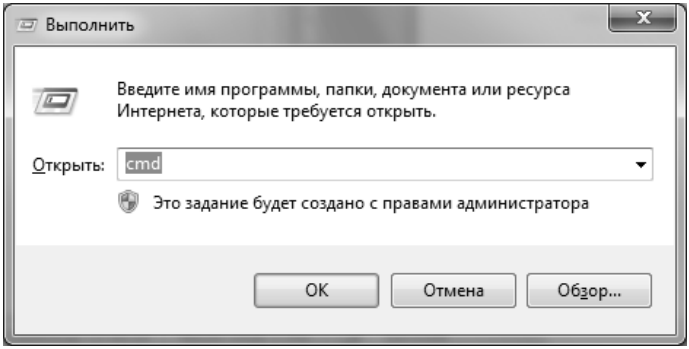

Рис. 1.4. Запуск командного інтерпретатора

У консолі командного інтерпретатора введіть команду *mingw32-make -v*. Якщо все зроблено правильно, то відобразиться інформація про утиліту *make* (рис. 1.5).

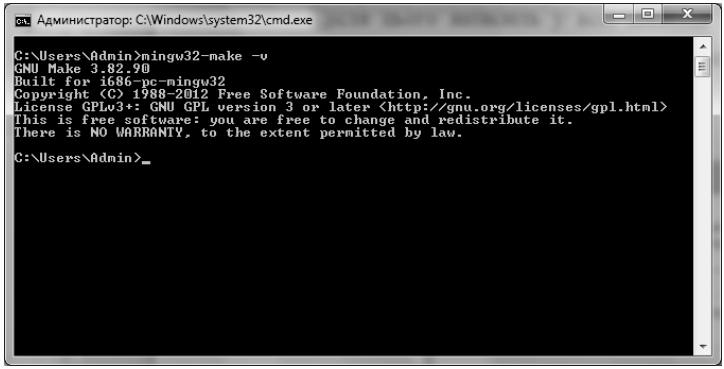

Рис. 1.5. Перевірка правильності налаштування змінних середовища

2. Інсталяція бібліотеки *wxWidgets*

Останню версію бібліотеки *wxWidgets* можна знайти за адресою: http://*wxWidgets*.org/downloads/. На момент написання посібника останньою стабільною версією була 3.0.2. Завантажте інсталяційний пакет або архів та встановіть його в певне місце файлової системи, наприклад на диск *C*. У нашому випадку шлях буде таким: *c:\wxWidgets-3.0.2*. Після інсталяції потрібно сконфігурувати бібліотеку для роботи в двох режимах – *debug* і *release*.

Запустимо консоль командного інтерпретатора (див. рис. 1.4) і перейдемо у ньому до підпапки *build\msw*. Після цього послідовно потрібно виконати такі команди:

1) *mingw32-make -f makefile.gcc CXXFLAGS="-std=gnu++11" BUILD=debug SHARED=0 MONOLITHIC=1 UNICODE=1* – для компіляції та зібрання бібліотек для відлагоджувального режиму;

2) *mingw32-make -f makefile.gcc CXXFLAGS="-std=gnu++11" BUILD=release SHARED=0 MONOLITHIC=1 UNICODE=1* – для компіляції та зібрання бібліотек для режиму реалізації.

Ці дві команди можуть виконуватися досить тривалий час (15–40 хв) залежно від потужності вашої машини. Якщо ваш ПК містить багатоядерний процесор, то швидкість компіляції та збирання можна збільшити, задавши перед *-f* ключ *-jn*, де *n* – кількість задач і кількість ядер процесора. Наприклад, попередня команда для відлагоджувального режиму на двоядерній машині буде такою: *mingw32-make -j2 -f makefile.gcc CXXFLAGS="-std=gnu++11" BUILD=debug SHARED=0 MONOLITHIC=1 UNICODE=1*.

Опція *CXXFLAGS="-std=gnu++11"* необхідна для успішної компіляції модулів бібліотеки, написаних із використанням C++11.

Тема 2
Структура додатка на базі бібліотеки *wxWidgets*

Теоретичні відомості

Клас додатка

Мінімальний додаток на *wxWidgets* показує головне вікно (клас wxFrame) із рядками меню та статусу. Меню дозволяє отримати інформацію «About program» або вийти з програми (рис. 2.1). Це дуже простий додаток, проте його цілком вистачає, щоб проілюструвати деякі базові принципи побудови додатків. Крім того, із накопиченням досвіду, можна використати цей приклад, додаючи в нього необхідну вам функціональність.

Рис. 2.1. Найпростіший додаток *wxWidgets*

Кожен додаток на *wxWidgets* визначає власний клас додатка, що є нащадком від базового класу *wxApp*. У програмі існує єдиний екземпляр цього класу, який є представленням цього додатка. Зазвичай цей клас перевизначає функцію-член `OnInit`, яка викликається, коли бібліотека *wxWidgets* готова виконати ваш код (функція є еквівалентом функції `main` мови *C* або `WinMain` у *Win32*-додатках). Мінімально можливе оголошення класу додатка наведено нижче (лістинг 2.1, файл `MyApp.h`).

```
#include <wx/app.h>
// Оголошуємо клас додатка
class MyApp : public wxApp{
public:
// Викликається при старті додатка
 virtual bool OnInit();
};
DECLARE_APP(MyApp)
```

Лістинг 2.1. Приклад оголошення класу додатка

Макрос `DECLARE_APP` вказує на клас, що містить точку входу до програми. Коли *wxWidgets* створює об'єкт типу `MyApp`, результат зберігається в глобальній змінній `wxTheApp`. Можна використати цю змінну в програмі, проте краще явно не звертатися до неї у своєму коді. Замість цього макрос `DECLARE_APP` дозволяє використати в програмі функцію `wxGetApp`, яка повертає посилання на об'єкт додатка типу `MyApp`.

Важливо! Навіть якщо не використовується макрос `DECLARE_APP`, усе одно можна використати змінну `wxTheApp` для доступу до функцій `wxApp`. Це дозволяє уникнути необхідності включення заголовочних файлів додатка.

Реалізація функції `OnInit` зазвичай створює принаймні одне вікно, прочитує параметри з командного рядка, ініціалізує потрібні для роботи структури даних і виконує інші дії, необхідні для запуску програми. Якщо функція повертає `true`, то *wxWidgets* запускає петлю подій, які реагують на дії користувача, і запускає відповідні обробники подій у разі потреби. Якщо ця функція повертає `false`, то *wxWidgets* коректно очищує свої внутрішні структури і завершує роботу додатка.

Проста реалізація функції `OnInit` може мати такий вигляд (лістинг 2.2, файл `MyApp.cpp`):

```
#include "MyApp.h"
#include "MyFrame.h"
IMPLEMENT_APP(MinimalWxApp)
// Ініціалізували додаток
bool MyApp::OnInit(){
// Створюємо головне вікно додатка
MyFrame *frame=new MyFrame(wxT("Minimal wxWidgets App"));
// Показуємо його
frame->Show(true);
// Запускаємо петлю подій
return true;
}
```

Лістинг 2.2. Приклад типової функції `OnInit`

Реалізація створює екземпляр нового класу вікна `MyFrame` (визначений пізніше), показує вікно на екрані та повертає `true`, щоб запустити петлю подій. Головні вікна (зокрема, фрейми й діалоги), на відміну від дочірніх, мають бути показані відразу після створення.

Заголовок головного вікна передається конструктору через макрос `wxT()`. Цей макрос використовується в більшості прикладів бібліотеки

wxWidgets і дає можливість перетворити рядки й символи до правильного типу, що дозволяє додатку компілюватися в режимі *Unicode*. Цей макрос є синонімом макросу `_T()`. Також у програмах можна зустріти макрос `_()`, який говорить бібліотеці, що рядок можна перекласти іншою мовою.

Де ж знаходиться код, який створює екземпляр класу MyApp? *wxWidgets* робить це самостійно, але потрібно явно вказати тип об'єкта, який необхідно створити. Для цього у файл із реалізацією потрібно додати інструкцію `IMPLEMENT_APP(MyApp)`. Без цього визначення *wxWidgets* не зможе створити об'єкт вашого додатка. Цей макрос також вставляє код, який перевіряє, що об'єкт є додатком, а бібліотека була скомпільована з необхідними опціями. Це дозволяє *wxWidgets* повідомляти про деякі помилки, які згодом могли б викликати появу помилок часу виконання, які важко виявити.

Клас фрейму

Розглянемо клас фрейму `MyFrame`. Фрейм є основним вікном, яке містить усі інші вікна і зазвичай має меню й рядок стану. Створимо новий файл заголовка й розмістимо в ньому такий код (лістинг 2.3, файл `MyFrame.h`):

```
#include <wx/menu.h>
#include <wx/frame.h>
#include <wx/statusbr.h>
class MyFrame : public wxFrame{// Клас головного вікна
public:
 MyFrame(const wxString& title); //Конструктор
 void OnQuit(wxCommandEvent& event); // Обробники подій
 void OnAbout(wxCommandEvent& event);
private:
 static const long idMenuQuit;
 static const long idMenuAbout;
 static const long ID_STATUSBAR1;
 DECLARE_EVENT_TABLE()// Декларація таблиці подій
};
```

Лістинг 2.3. Оголошення класу `MyFrame`

Клас для фрейму містить конструктор, два обробники подій (меню, що зв'язують пункти з C++ кодом) і макрос, який повідомляє, що клас містить обробники подій.

Обробники подій

Функції обробки подій у `MyFrame` не є віртуальними і не мають бути віртуальними. Тоді як же вони викликаються? Для цього у файлі

реалізації потрібно визначити таблицю обробників подій (лістинг 2.4, файл `MyFrame.cpp`):

```
#include "MinimalWxFrame.h"
#include <wx/msgdlg.h>
#include <wx/string.h>
const long MinimalWxFrame::idMenuQuit = wxNewId();
const long MinimalWxFrame::idMenuAbout = wxNewId();
const long MinimalWxFrame::ID_STATUSBAR1 = wxNewId();
//Таблиця подій для MyFrame
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
 EVT_MENU(idMenuAbout, MyFrame::OnAbout)
 EVT_MENU(idMenuQuit, MyFrame::OnQuit)
END_EVENT_TABLE()
```

Лістинг 2.4. Таблиця подій для класу `MyFrame`

У вказаній вище таблиці натиснення кнопки миші на пунктах меню з ідентифікаторами `idMenuQuit` і `idMenuAbout` спрямовується функціям `MyFrame::OnQuit` і `MyFrame::OnAbout` відповідно. Макрос `EVT_MENU` є одним із багатьох можливих макросів таблиці подій, які можна використати, щоб вказати *wxWidgets*, які типи подій направляти у функцію. Ідентифікатори меню створюються функцією `wxNewId` під час ініціалізації додатка, однак їх можна визначити за допомогою перерахувань (`enum`), констант або директив препроцесора (`#define`).

Однак *wxWidgets* дозволяє динамічно, тобто під час виконання, призначати обробники подій тим чи іншим компонентам. Робиться це за допомогою функції `Connect`, яка сприймає три параметри – ідентифікатор об'єкта, тип події для компонента та об'єкт типу `wxObjectEventFunction`, який є ні чим іншим, як функцією-обробником відповідної події. Наприклад, замість статичного визначення обробників у конструкторі фрейму можна додати такі інструкції:

```
Connect(idMenuQuit,wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction)&MyFrame::OnQuit);
Connect(idMenuAbout,wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction)&MyFrame::OnAbout);
```

Розглянемо тепер обробники подій (лістинг 2.5, файл `MyFrame.cpp`):

```
void MyFrame::OnAbout(wxCommandEvent& event){
 wxString msg;
 msg.Printf(wxT("Hello and welcome to %s"),
```

```
            wxVERSION_STRING);
    wxMessageBox(msg, wxT("About Minimal"),
                 wxOK | wxICON_INFORMATION, this);
}
void MyFrame::OnQuit(wxCommandEvent& event)
{
  // Знищуємо фрейм
  Close();
}
```

Лістинг 2.5. Обробники подій натискання на пункти меню

Метод `MyFrame::OnAbout` показує невелике повідомлення користувача, коли той вибирає з меню пункт *About*. Функція `wxMessageBox` приймає як аргументи текст повідомлення, заголовок вікна, комбінацію стилів і вказівку на батьківське вікно.

Як вказано в таблиці подій, функція `MyFrame::OnQuit` викликається, коли користувач вибирає в меню пункт Quit2. В обробнику цієї події явно викликається метод Close для знищення фрейму й завершення роботи додатка, оскільки програма складається всього з одного фрейму. Насправді метод `Close` не знищує додаток, він просто генерує подію `wxEVT_CLOSE_WINDOW`, обробник якої за замовчуванням знищує фрейм, використовуючи для цього метод `wxWindow::Destroy`.

Існують інші шляхи для завершення роботи додатка – користувач може клікнути на кнопці закриття, розташованій на фреймі, або вибрати пункт `Close` із системного меню (чи меню віконного менеджера середовища запуску). У цих випадках метод фрейму `OnQuit` не викликається, а *wxWidgets* посилає фрейму подію `wxEVT_CLOSE_WINDOW` через виклик функції `Close` (подібно до того, як це робиться в методі `OnQuit`). *wxWidgets* за замовчуванням перехоплює цю подію і знищує вікно. Додаток може перевизначити цю поведінку і, наприклад, просити підтвердження в користувача перед закриттям.

У нашому прикладі цього робити не вимагалося, але велика частина додатків реалізує функцію `OnExit` у класі додатка для коректного очищення структур даних перед виходом. Зверніть увагу, що ця функція викликається, тільки якщо `OnInit` повернула `true`.

Наприкінці розглянемо конструктор фрейму, у якому ініціалізується іконка для додатка, меню і рядок стану (лістинг 2.6, файл `MyFrame.cpp`):

```
#include "mondrian.xpm"
MyFrame::MyFrame(const wxString& title)
```

```
        : wxFrame(NULL, wxID_ANY, title){
    // Встановлюємо іконку для фрейму
    SetIcon(wxIcon(mondrian_xpm));
    // Створюємо меню
    wxMenu *fileMenu = new wxMenu;
    // Додаємо пункт "About"
    wxMenu *helpMenu = new wxMenu;
    helpMenu ->Append(idMenuAbout, wxT("&About...\tF1"),
                      wxT("Show about dialog"));
    fileMenu ->Append(idMenuQuit, wxT("E&xit\tAlt - X"),
                     wxT("Quit this program"));
    //Додаємо створене меню в рядок меню...
    wxMenuBar *menuBar = new wxMenuBar();
    menuBar->Append(fileMenu, wxT("&File"));
    menuBar ->Append(helpMenu, wxT("&Help"));
    //і приєднуємо до фрейму
    SetMenuBar(menuBar);
    // Створюємо рядок стану для краси
    CreateStatusBar(2);
    SetStatusText(wxT("Welcome to wxWidgets"!));
}
```

Лістинг 2.6. Конструктор фрейму

Залежно від платформи запускається функція `main`, `WinMain` або еквівалентна їй (ця функція надається бібліотекою *wxWidgets*, а не додатком). *wxWidgets* ініціалізує внутрішні структури даних і створює екземпляр класу `MyApp`.

Далі *wxWidgets* викликає метод MyApp::`OnInit`, який створює екземпляр класу `MyFrame`.

Конструктор класу `MyFrame` створює вікно через конструктор `wxFrame` і додає іконку, меню й рядок стану.

Метод `MyApp::OnInit` показує фрейм і повертає `true`.

wxWidgets запускає петлю подій, чекає чергової події та викликає обробник, визначений для цієї події.

Як уже було вказано, додаток завершується, коли закривається основний фрейм, що можна зробити через відповідний пункт меню, через стандартні кнопки або стандартні меню (це залежить від платформи).

Практична робота

Розширимо наші знання про *wxWidgets*, створивши простий додаток для обчислення коренів квадратних рівнянь. Завантажимо Code::Blocks і створимо новий проект командою *File >> New >> Project*... У діалозі (рис. 2.1) виберемо *wxWidgets projects* і натиснемо **Go**.

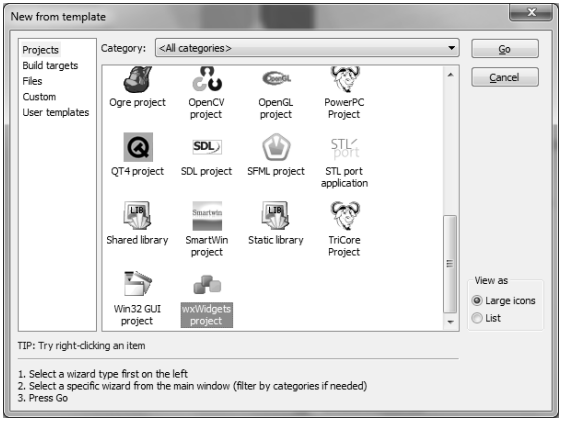

Рис. 2.1. Діалог вибору проекту

Наступним може з’явитися діалог із додатковою інформацією. Для того, щоб він не з’являвся наступного разу, установіть прапорець *Skip* і натисніть *Next*.

У наступному діалозі виберемо версію *wxWidgets* і натиснемо *Next* (рис. 2.2).

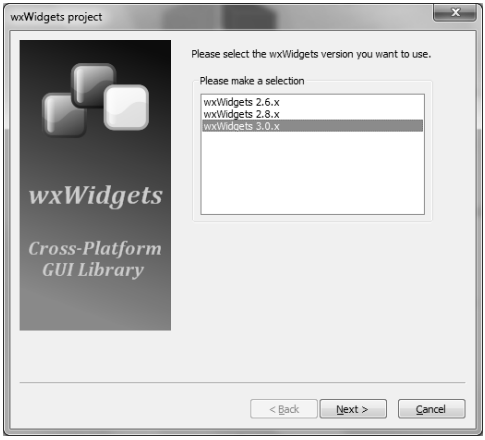

Рис. 2.2. Діалог вибору версії *wxWidgets*

Наступний діалог потребує вказати розташування файлів проекту. Уведемо в перше поле *wxSquareRootEx*, а в другому вкажемо папку *C:\Temp* і натиснемо *Next* (рис. 2.3).

Наступний діалог дає можливість увести інформацію про розробника цього додатка (рис. 2.4).

Натиснувши *Next*, перейдемо до діалогу, де потрібно обрати дизайнер користувацького інтерфейсу (відсутній, *wxSmith* – плагін для Code::Blocks, *wxFormBuilder* – окремий додаток, який можна завантажити на сайті http://sourceforge.net/projects/wxformbuilder). Також потрібно обрати тип графічного інтерфейсу – діалоговий (*Dialog* `Based`) або звичайний (*Frame* `Based`) (рис. 2.5). Виберемо *None* і *Frame* `Based` та натиснемо *Next*.

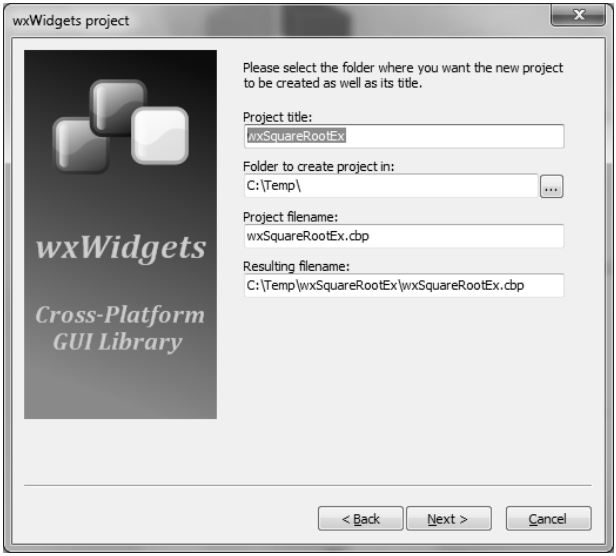

Рис. 2.3. Діалог для вказування імені та розташування проекту

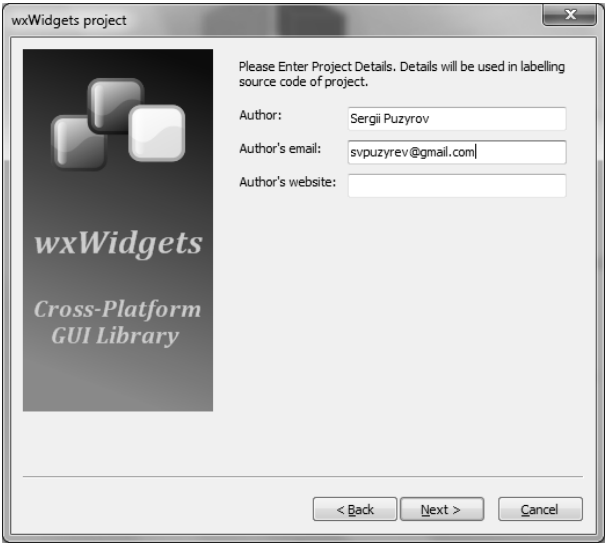

Рис. 2.4. Діалог для інформації про автора проекту

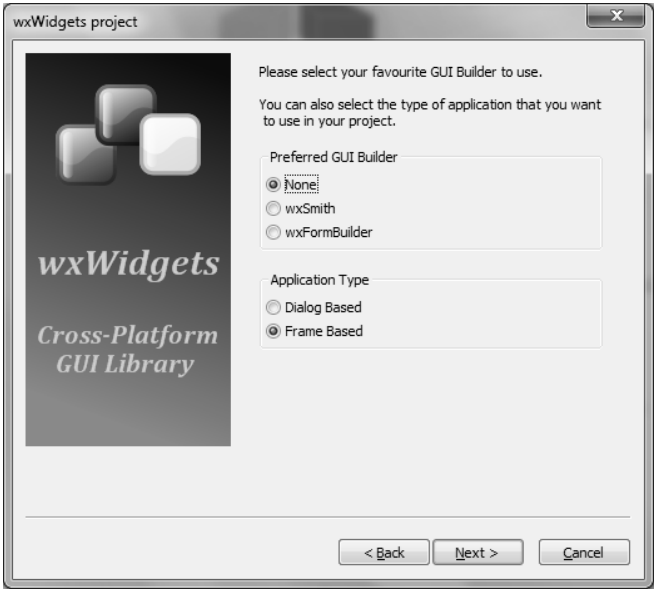

Рис. 2.5. Тип інтерфейсу користувача

У наступному вікні можна вказати шлях до файлів бібліотеки *wxWidgets* (рис. 2.6).

Рис. 2.6. Шлях до бібліотеки *wxWidgets*

Змінна *wx* містить шлях до *wxWidgets* і може бути визначена командою меню *Settings >> Global variables*.

Натиснувши *Next*, обираємо створення двох реалізацій проекту (рис. 2.7).

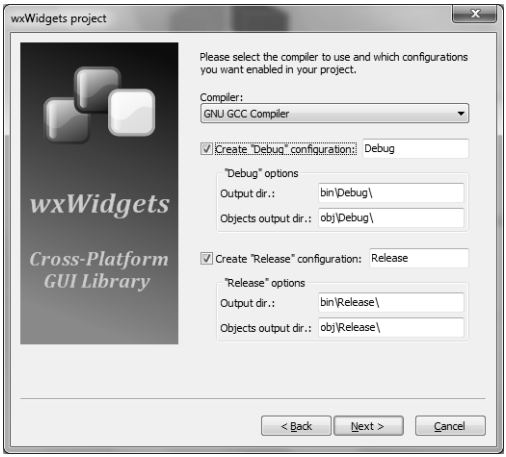

Рис. 2.7. Конфігурації проекту

Після натискання *Next* у наступному діалозі відмічаємо прапорці, показані на рис. 2.8.

Рис. 2.8. Конфігурація *wxWidgets*

Прапорці *wxWidgets is built as a monolithic library* та *Enable unicode* мають бути встановлені, якщо бібліотека компілювалася з параметрами MONOLITHIC=1 та UNICODE=1 (див. практичну роботу № 1).

Натиснувши *Next*, залишаємо всі опції за замовчуванням і натискаємо *Finish* (рис. 2.9).

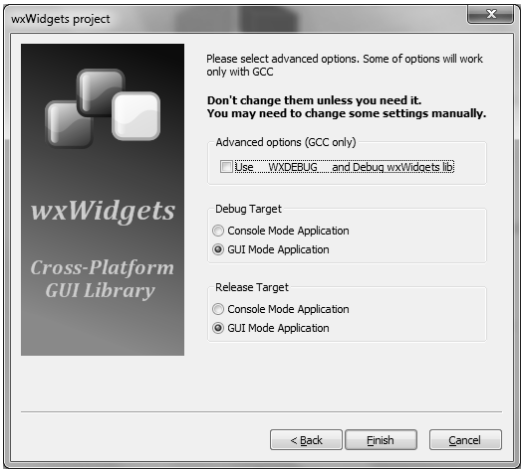

Рис. 2.9. Завершення роботи майстра

Отримаємо такий вигляд середовища Code::Blocks (рис. 2.10)

Далі створимо клас додатка. Для цього виконаємо команду *File* >> `New` >> *Class* і в полі *Class name* введемо *wxSquareRootApp* і натиснемо ***Create*** (рис. 2.11).

У діалогах, що з'являться, натиснемо ***Yes*** і ***Ok***.

Аналогічно додамо клас фрейму, назвавши його *wxSquareFrame.*

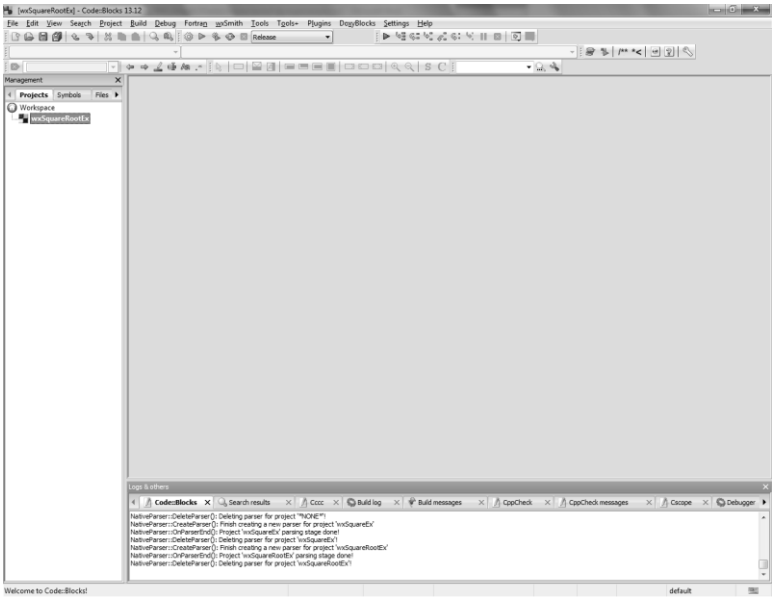

Рис. 2.10. Середовище Code::Blocks

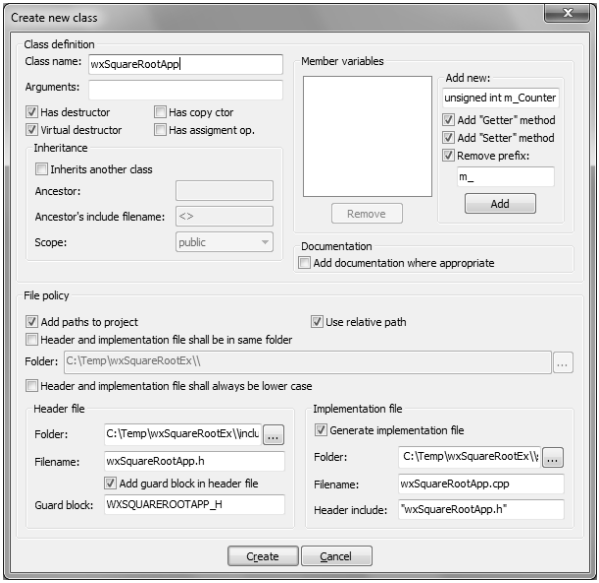

Рис. 2.11. Діалог створення класу додатка

Відредагуємо файл *wxSquareRootApp.h*, додавши заголовочний файл бібліотеки, перевизначивши `OnInit` та додавши макрос `DECLARE_APP`. Також додамо реалізацію `OnInit` та макрос `IMPLEMENT_APP` у файл реалізації *wxSquareRootApp.cpp* (лістинг 2.7).

```
// wxSquareRootApp.h
#ifndef WXSQUAREROOTAPP_H
#define WXSQUAREROOTAPP_H
#include <wx/app.h>
class wxSquareRootApp:public wxApp{
public:
 wxSquareRootApp();
 virtual ~wxSquareRootApp();
 bool virtual OnInit();
 protected:
 private:
};
DECLARE_APP(wxSquareRootApp)
#endif // WXSQUAREROOTAPP_H
#include "wxSquareRootApp.h"
#include "wxSquareRootFrame.h"
// wxSquareRootApp.cpp
```

```
IMPLEMENT_APP(wxSquareRootApp)
wxSquareRootApp::wxSquareRootApp(){}
wxSquareRootApp::~wxSquareRootApp(){}

bool wxSquareRootApp::OnInit(){
 wxString title = wxT("Square root solver");
 wxSquareRootFrame* frame = new wxSquareRootFrame(title);
 frame->Show();
 return true;
}
```

Лістинг 2.7. Файли декларації та реалізації класу додатка

Важливо! Зверніть увагу, що клас додатка є нащадком від класу `wxApp`.

Аналогічно відредагуємо заголовочний файл фрейму. Додамо заголовочні файли меню, статичного тексту, текстового елемента управління.

Визначимо також статичні ідентифікатори для трьох пунктів меню:

- `idMenuCalculate` – запускає процес обчислень;
- `idMenuQuit` – завершує роботу додатка;
- `idMenuAbout` – виводить інформацію про додаток.

Додамо також об'явлення відповідних цим пунктам меню обробників: `OnCalculate`, `OnQuit`, `OnAbout`.

Додамо закриті поля, які будуть вказувати на текстові мітки та текстові поля для елементів управління. У лістингу 2.8 наведено зміст файлу декларації фрейму:

```
#ifndef WXSQUAREROOTFRAME_H
#define WXSQUAREROOTFRAME_H

#include <wx/frame.h>
#include <wx/menu.h>
#include <wx/menuitem.h>
#include <wx/statusbr.h>
#include <wx/stattext.h>
#include <wx/textctrl.h>
class wxSquareRootFrame:public wxFrame{
public:
 wxSquareRootFrame(wxString title);
 virtual ~wxSquareRootFrame();
protected:
private:
 wxStaticText* ACoefLabel;
 wxStaticText* BCoefLabel;
```

```
    wxStaticText* CCoefLabel;
    wxStaticText* ResultLabel;
    wxTextCtrl* ACoefText;
    wxTextCtrl* BCoefText;
    wxTextCtrl* CCoefText;
    wxMenuBar* MenuBar1;
    wxStatusBar* StatusBar;
    static const long idMenuCalculate;
    static const long idMenuQuit;
    static const long idMenuAbout;
    static const long idStatusBar;
    static const long idLabelACoef;
    static const long idLabelResult;
    static const long idTextACoef;
    static const long idLabelBCoef;
    static const long idTextBCoef;
    static const long idLabelCCoef;
    static const long idTextCCoef;
    void OnCalculate();
    void OnQuit();
    void OnAbout();

    DECLARE_EVENT_TABLE()
};
#endif // WXSQUAREROOTFRAME_H
```

Лістинг 2.8. Файл декларації класу фрейму

Реалізуємо тепер конструктор фрейму та обробники відповідних подій (лістинг 2.9):

```
#include "wxSquareRootFrame.h"
#include <wx/msgdlg.h>
#include <cmath>
const long wxSquareRootFrame::idMenuCalculate= wxNewId();
const long wxSquareRootFrame::idMenuQuit=wxNewId();
const long wxSquareRootFrame::idMenuAbout=wxNewId();
const long wxSquareRootFrame::idStatusBar=wxNewId();

const long wxSquareRootFrame::idLabelACoef=wxNewId();
const long wxSquareRootFrame::idLabelBCoef=wxNewId();
const long wxSquareRootFrame::idLabelCCoef=wxNewId();
const long wxSquareRootFrame::idLabelResult=wxNewId();

const long wxSquareRootFrame::idTextACoef=wxNewId();
const long wxSquareRootFrame::idTextBCoef=wxNewId();
const long wxSquareRootFrame::idTextCCoef=wxNewId();
```

```
BEGIN_EVENT_TABLE(wxSquareRootFrame,wxFrame)
END_EVENT_TABLE()

wxSquareRootFrame::wxSquareRootFrame(wxString title)
                  : wxFrame(NULL, wxID_ANY, title){
 wxMenuItem* MenuItemCalculate;
 wxMenuItem* MenuItemQuit;
 wxMenuItem* MenuItemAbout;
 wxMenu* MenuFile = new wxMenu();
 wxMenu* MenuHelp = new wxMenu();
 MenuBar1 = new wxMenuBar();
 MenuItemCalculate = new wxMenuItem(MenuFile,
              idMenuCalculate, wxT("&Calculate\tCtrl-C"),
              wxT("Calculate roots"));
 MenuFile->Append(MenuItemCalculate);
 MenuFile->AppendSeparator();
 MenuItemQuit = new wxMenuItem(MenuFile,idMenuQuit,
                            wxT("&Quit\tAlt-F4"),
                            wxT("Close application"));
 MenuFile->Append(MenuItemQuit);

 MenuItemAbout = new wxMenuItem(MenuFile,idMenuAbout,
        wxT("&About"), wxT("Help about application"));
 MenuHelp->Append(MenuItemAbout);
 MenuBar1->Append(MenuFile,_T("&File"));
 MenuBar1->Append(MenuHelp,_T("&Help"));
 SetMenuBar(MenuBar1);
 StatusBar = new wxStatusBar(this,idStatusBar,0,
                             wxT("idStatusBar"));
 int _statusBarWidths[1]={-1};
 int _statusBarStyles[1]={wxSB_NORMAL};
 StatusBar->SetFieldsCount(1,_statusBarWidths);
 StatusBar->SetStatusStyles(1,_statusBarStyles);
 SetStatusBar(StatusBar);
 Connect(idMenuCalculate, wxEVT_COMMAND_MENU_SELECTED,
  (wxObjectEventFunction)&wxSquareRootFrame::OnCalculate);
 Connect(idMenuQuit, wxEVT_COMMAND_MENU_SELECTED,
  (wxObjectEventFunction)&wxSquareRootFrame::OnQuit);
 Connect(idMenuAbout, wxEVT_COMMAND_MENU_SELECTED,
  (wxObjectEventFunction)&wxSquareRootFrame::OnAbout);
 ACoefLabel = new wxStaticText(this,idLabelACoef,
             wxT("A:"), wxPoint(50,43),
             wxSize(20,20), 0, wxT("idLabelACoef"));
 ACoefText = new wxTextCtrl(this,idTextACoef, wxT("1"),
              wxPoint(70,40), wxSize(50,20), 0,
```

```
                wxDefaultValidator, wxT("idTextACoef"));
 BCoefLabel = new wxStaticText(this,idLabelBCoef,
               wxT("B:"), wxPoint(130,43),
               wxSize(20,20),0, wxT("idLabelBCoef"));
 BCoefText = new wxTextCtrl(this,idTextBCoef,
          wxT("1"), wxPoint(150,40), wxSize(50,20), 0,
          wxDefaultValidator,   wxT("idTextBCoef"));
 CCoefLabel = new wxStaticText(this,idLabelCCoef,
          wxT("C:"), wxPoint(205,43), wxSize(20,20),0,
          wxT("idLabelCCoef"));
 CCoefText = new wxTextCtrl(this,idTextCCoef,
          wxT("1"), wxPoint(225,40), wxSize(50,20), 0,
          wxDefaultValidator, wxT("idTextCCoef"));
 ResultLabel = new wxStaticText(this,idLabelResult,
              wxT("Result:"),wxPoint(50,80),
              wxSize(50,20), 0, wxT("idLabelResult"));
}

wxSquareRootFrame::~wxSquareRootFrame(){}

void wxSquareRootFrame::OnCalculate(){
 double a=1,b=1,c=1;
 double x1=0,x2=0;
 wxString result=wxT("");
 if (ACoefText->GetValue().ToDouble(&a)&&
  BCoefText->GetValue().ToDouble(&b)&&
  CCoefText->GetValue().ToDouble(&c)){
   double D=b*b-4*a*c;
   if (D<0){
    result<<"Roots don't exist";
   }
   else{
    if (D==0){
     x1=x2=-b/2/a;
     result<<"One root: "<<x1;
    }
    else{
     x1=(-b-sqrt(D))/2/a;
     x2=(-b+sqrt(D))/2/a;
     result<<"Roots: x1="<<x1<<"; x2="<<x2;
    }
   }
  }
  ResultLabel->SetLabel(result);
}
void wxSquareRootFrame::OnQuit(){
```

```
 Close();
}
void wxSquareRootFrame::OnAbout(){
 wxString aboutString = wxT("wxSquareRoot\
  application solves square equations");
 wxMessageBox(aboutString);
}
```

Лістинг 2.9. Файл реалізації класу фрейму

Після компіляції та запуску отримаємо вікно програми з пунктами меню та текстовими полями. Увівши в них деякі числа та виконавши команду Calculate, отримаємо такий результат (рис. 2.12):

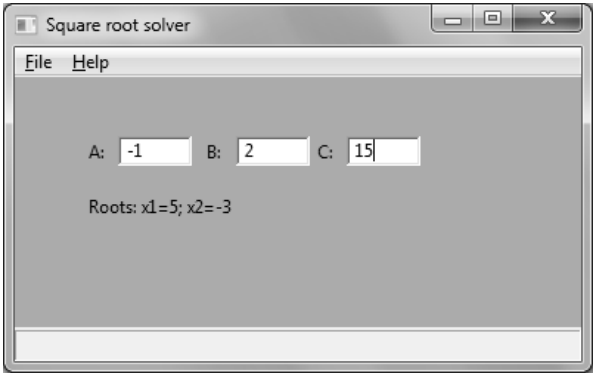

Рис. 2.12. Програма для обчислення квадратних коренів

Тема 3
Робота з дизайнером *wxSmith*

Теоретичні відомості

Дизайнер дозволяє значно прискорити проектування користувацького інтерфейсу шляхом візуального моделювання фрейму з розміщеними на ній компонентами.

Компоненти дизайнера стають доступними, якщо на етапі створення проетку *wxWidgets* вибрати тип дизайнера *wxSmith* (рис. 3.1).

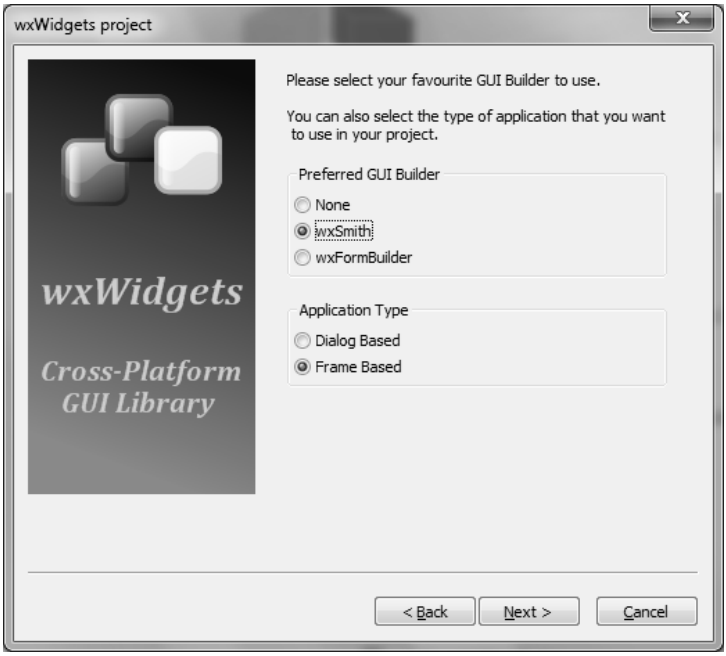

Рис. 3.1. Вибір дизайнера інтерфейсу

Також не потрібно встановлювати прапорець *Create Empty Project* (рис. 3.2). Інші етапи створення аналогічні розглянутим вище.

На рис. 3.3 представлено вигляд *Code::Blocks* із дизайнером форми.

Рис. 3.2. Створення шаблону додатка з GUI

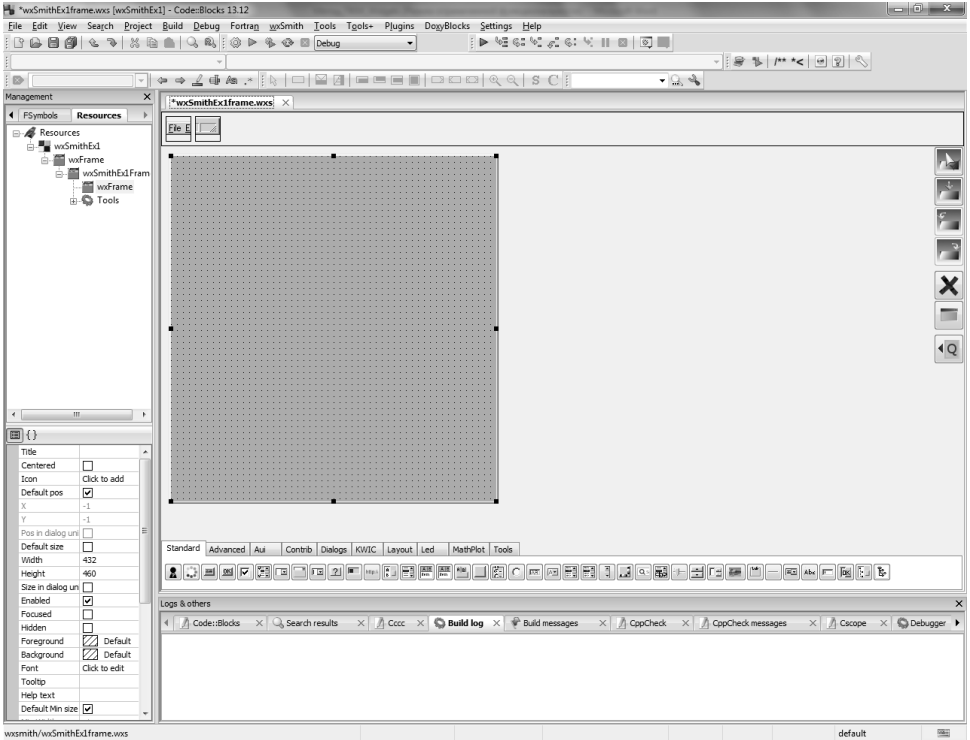

Рис. 3.3. Шаблон додатка з GUI

Дизайнер складається з трьох частин: редактора меню, шаблону вікна додатка та панелей із компонентами користувацького інтерфейсу.

Якщо потрібно додати пункт меню, треба двічі клікнути на ньому й у діалозі задати потрібні налаштування.

Компоненти на форму поміщуються шляхом перетягування. Якщо фокус знаходиться в певного елемента управління, то в лівій нижній частині дизайнера відображаються його властивості. Також там можна задавати обробники подій для певного елемента управління.

Кожний компонент представлений у *wxWidgets* відповідним класом. Наприклад, текстове поле – це клас `wxTextCtrl`, мітка (статичний текст) – `wxStaticText`, кнопка – `wxButton`, прапорець та список прапорців – `wxCheckBox` та `wxCheckListBox`, список рядків – `wxListBox`, радіокнопка та їх список – відповідно `wxRadioButton` і `wxRadioBox`. Усі ці та інші компоненти розміщені на вкладці *Standart*.

Практична робота

Створимо додаток із GUI, який знаходить усі прості числа в заданому діапазоні. Для початку визначимося з елементами управліннння:

- два текстові поля й відповідні мітки для них, у які вводяться мінімальне та максимальне значення діапазону;
- список, який буде містити всі знайдені прості числа.

Також додамо пункт меню, який буде запускати процес пошуку простих чисел. Для цього в дизайнері двічі клікнемо на меню. Відкриється діалог (рис. 3.4), у якому перейдемо до меню *&File* і виділимо *Quit*, натиснемо кнопку ***New*** і заповнимо поля, як показано на рисунку, і натиснемо стрілку ^ для переміщення меню вище. Потім знову натиснемо ***New***, але виберемо опцію *Separator* і натиснемо *OK*.

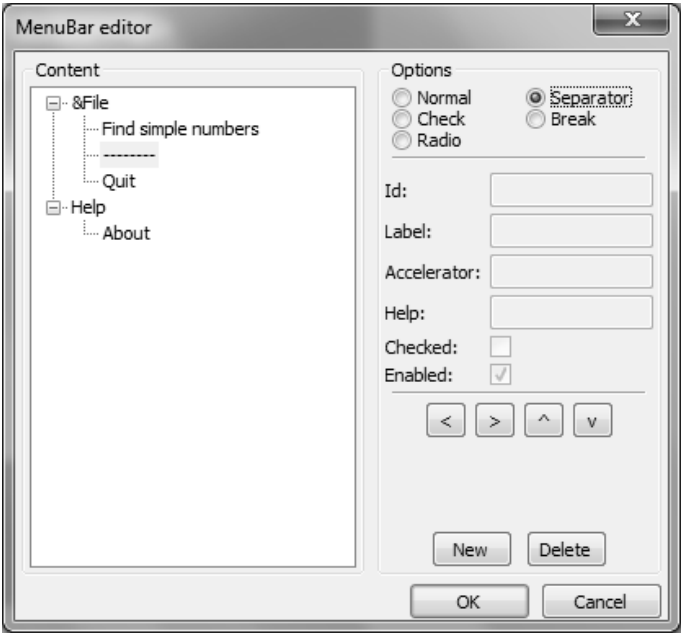

Рис. 3.4. Редактор меню

Розмістимо тепер елементи статичний текст і текстове поле, а також список рядків і кнопку (рис. 3.5)

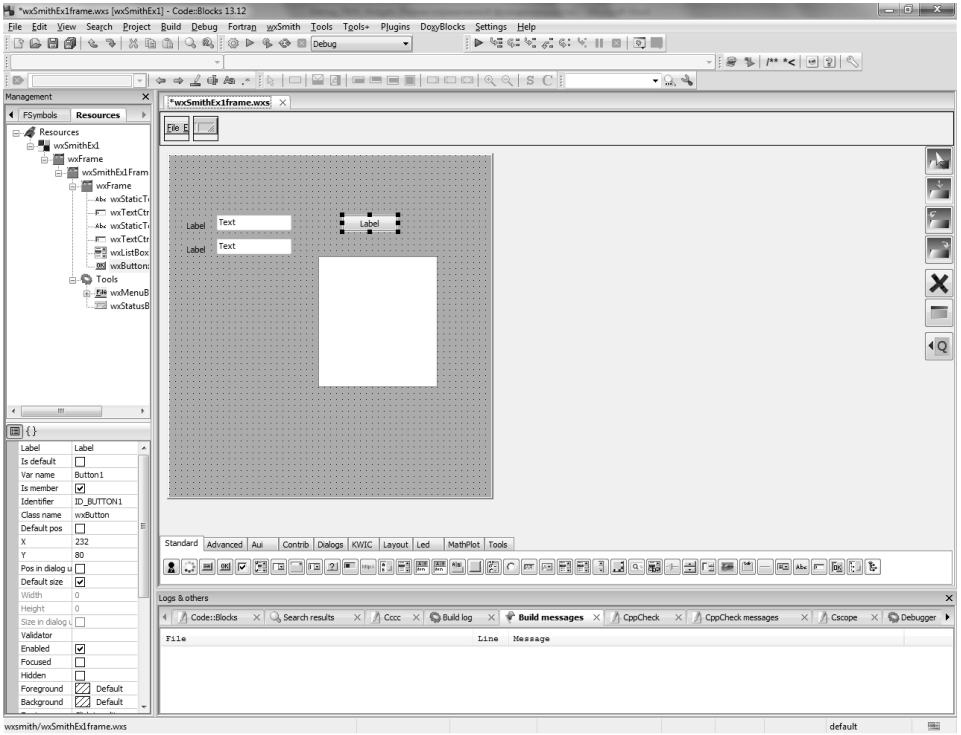

Рис. 3.5. Форма з компонентами

Дамо осмислені назви нашим елементам управління. Для цього виділяємо по черзі кожний із них, у редакторі їхніх властивостей задаємо:

– для StaticText1 – Label=Low number, Var name=LowNumberLabel, Identifier=ID_LowNumberLabel;

– для StaticText2 – Label=High number, Var name=HighNumberLabel, Identifier= ID_ HighNumberLabel;

– для TextCtrl1 – Text=0, Var name=LowNumberText, Identifier= ID_LowNumberText;

– для TextCtrl2 – Text=1, Var name=HighNumberText, Identifier= ID_HighNumberText;

– для Button1 – Label=Find, Var name=FindButton, Identifier= ID_FindButton;

– для ListBox1 – Var name=SimpleNumberList, Identifier= ID_SimpleNumberList.

Перейдемо до файлу заголовка форми й додамо туди захищений метод:

```
bool findSimpleNumbers(),
```

а у файлі реалізації – його тіло, представлене в лістингу 3.1:

```
bool wxSmithEx1Frame::findSimpleNumbers(){
 long startN=0,stopN=1;
 if (LowNumberText->GetValue().ToLong(&startN) &&
  HighNumberText->GetValue().ToLong(&stopN)){
  SimpleNumberList->Clear();
  for (long i=startN;i<=stopN;++i){
   bool isSimple=true;
   for (long j=2;j<i && isSimple;++j){
    isSimple=i%j!=0;
   }
   if (isSimple){
    wxString line=wxT("");
    line<<i;
    SimpleNumberList->AppendAndEnsureVisible(line);
   }
  }
  return SimpleNumberList->GetCount()>0;
 }
 else{
  return false;
 }
}
```

Лістинг 3.1. Функція пошуку простих чисел

Далі двічі клікнемо на кнопці й відразу згенеруємо обробник події для кнопки, куди й помістимо виклик цієї функції:

```
void wxSmithEx1Frame::OnFindButtonClick(
                          wxCommandEvent& event)
{
 findSimpleNumbers();
}
```

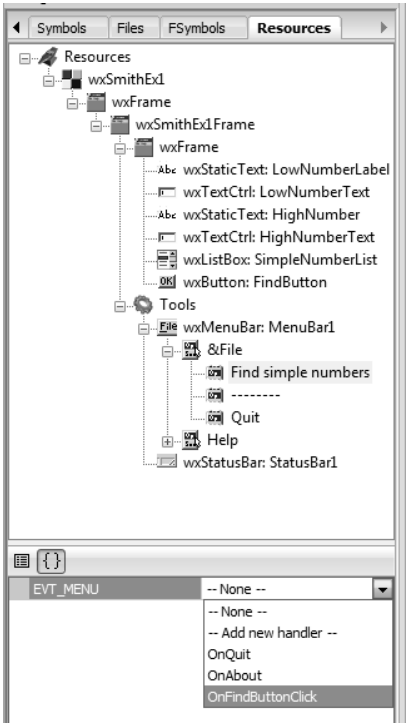

Рис. 3.6. Обробник для пункту меню

Аналогічно згенеруємо обробник події для пункту меню *Find simple numbers*. Для цього в дизайнері виділимо цей пункт і перейдемо на вкладку подій, натиснувши {} (рис. 3.6), і виберемо `OnFindButtonClick`. У згенерований метод помістимо також виклик функції `findSimpleNumbers()`.

Тема 4
Рядки, масиви, списки, відображення та дати в *wxWidgets*

Теоретичні відомості

Клас `wxString`

wxWidgets використовує власний строковий клас `wxString`, який використовується як усередині бібліотеки, так і для передачі параметрів і повернення результату. `wxString` підтримує майже всі стандартні операції, які потрібні строковому класу: динамічне управління пам'яттю, створення з *C* рядків і символів, оператор присвоювання, доступ до окремих символів, об'єднання та порівняння рядків, виділення підрядків, перетворення регістру, видалення символів, операції пошуку й заміни, аналоги функції `printf` тощо.

Попри те, що `wxString` є ще одним строковим класом, він підтримує деякі додаткові корисні можливості. `wxString` повністю підтримує юнікод, а також включає методи з перетворення рядка в різні кодові таблиці.

Використання `wxString` дає можливість передавати рядки в бібліотеку або отримувати назад без додаткового процесу перетворення. Крім того, `wxString` реалізує 90 % методів стандартного класу `std::string`. Більш детальну інформацію про методи можна отримати з документації щодо *wxWidgets*.

Символи й символьні літерали

wxWidgets використовує тип `wxChar`, який є синонімом типів `char` або `wchar_T`, залежно від типу збірки (ANSI або юнікод). У *wxWidgets* не потрібно розділяти тип для рядків `char` або `wchar_T`, оскільки `wxString` зберігає рядок, використовуючи відповідний тип. Тому працюючи безпосередньо з рядками `wxString`, бажано використати тип `wxChar` замість `char` або `wchar_T`. Дотримання цієї рекомендації дає гарантії, що ваш код працюватиме в ANSI та юнікодних збірках без використання заплутаних директив препроцесора.

Якщо використовувати *wxWidgets* в юнікодному режимі, то стандартні строкові літерали мають неправильний тип (`char*`). Щоб мати можливість використати літерали в юнікодному режимі, їх необхідно перетворити в константи з широкими символами, що зазвичай робиться за

допомогою спеціальної директиви L, яка вказується перед літералом. *wxWidgets* вводить макрос wxT (що є синонімом _T), щоб отримувати правильний літерал незалежно від поточного режиму збірки. Якщо юнікодний режим не використовується, то _T перетвориться на порожній макрос, але під час збирання в юнікодному режимі додається необхідний символ L для перетворення в літерал із широкими символами. Наприклад:

```
wxChar ch=wxT('*');
wxString s = wxT("Hello, world"!);
wxChar* pChar = wxT("My string");
```

Іноді необхідно отримати доступ до даних wxString для низькорівневої обробки мовою *C*. Для цього в *wxWidgets* є декілька методів:

- mb_str повертає рядок мови *C* із типом const char*. У юнікодному режимі для цього відбувається відповідне перетворення, тому можуть бути втрачені деякі дані.
- wc_str повертає широкосимвольне представлення рядка з типом wchar_t*. В ANSI-режимі рядок для цього перетворюється в юнікод.
- c_str повертає вказівку на реальні типи рядків (типу const char* у режимі ANSI і const wchar_t* у юнікодному режимі). Ніякого перетворення при цьому не відбувається.

Можна виконувати перетворення між std::string і wxString, використовуючи метод c_str, як показано нижче:

```
std::string str1 = wxT("hello");
wxString str2 = str1.c_str();
std::string str3 = str2.c_str();
```

Однією з типових помилок у використанні wxString є надія на неявне перетворення wxString в const char *. Для її уникнення треба використовувати метод c_str.

Оскільки більшість програм активно використовує символьні рядки, то стандартна бібліотека мови *C* містить безліч функцій для роботи з ними. Проте не всі з них мають інтуїтивну поведінку (наприклад, функція strncpy не завжди ставить завершальний нуль у результуючий рядок) або є безпечними з точки зору можливості переповнення буфера. До того ж деякі корисні функції не є частиною стандарту. Тому на додаток до звичайних функцій класу wxString існує декілька глобальних функцій, визначених у заголовочному файлі "wx/string.h":

- wxIsEmpty перевіряє, чи є рядок порожнім (повертає true для вказівки NULL);

– wxStrlen коректно обробляє NULL і повертає для нього 0;

– wxStricmp, яка є платформонезалежною чутливою до регістра функцією порівняння, аналогічна функціям stricmp або strcasecmp.;

– wxSnprintf і wxVsnprintf, які може бути використано замість потенційно небезпечної стандартної функції sprintf. Ці функції використовують snprintf, яка в більшості випадків перевіряє розміри буфера. Також можна використати wxString::Printf, не турбуючись про недоліки стандартної функції printf.

Перетворення чисел та форматування рядків

Операції перетво-рення чисел на рядки та навпаки, як і комбінування рядків із числами, є типовими, тому не дивно, що wxString містить методи для цих операцій.

1. Метод ToLong(long* val, int base=10) намагається перетворити рядок на число зі знаком у вказаній системі числення. Метод повертає true в разі успіху (результат операції буде записано в змінну, на яку вказує val), або false, якщо рядок не представляється у вигляді коректного числа у вибраній системі числення base. Основа системи числення (base) може набувати значень від 2 до 36 включно або 0, яке означає використання стандартних правил мови *C*: якщо число починається з 0x, то воно є шістнадцятковим; якщо починається з 0, то вважається вісімковим, і десятковим у решті випадків.

2. Метод ToULong(unsigned long* val, int base=10) працює аналогічно ToLong, але перетворює рядок на беззнакове число.

3. Метод ToDouble(double* val) намагається перетворити рядок на число з плаваючою точкою. Повертається true в разі успіху (результат буде записано в змінну, на яку вказує val) або false, якщо рядок не є таким числом.

4. Метод Printf(const wxChar* pszFormat, ...) схожий на стандартну функцію мови *C* sprintf і дозволяє помістити інформацію в wxString, використовуючи стандартне форматування для рядків мови *C*. Метод повертає кількість записаних байт.

5. Статичний метод static Format(const wxChar* pszFormat, ...) повертає рядок wxString, що містить результат виклику Printf із параметрами.

Перевагою Format над Printf є можливість додавання результату Format до існуючого рядка, наприклад:

```
int n = 10;
wxString s = "Some Stuff";
s += wxString::Format(wxT("%d"), n );
```

6. Метод `operator <<` може використовуватися для додавання значень типу `int`, `float` або `double` до рядка `wxString`.

Розбиття на лексеми за допомогою класу `wxStringTokenizer`

`wxStringTokenizer` допоможе розбити рядок на декілька лексем, замінюючи та розширюючи можливості функції `strtok` мови *C*. Для цього необхідно створити об'єкт класу `wxStringTokenizer`, передати йому рядок, що вимагає розбиття, і роздільник лексем (за замовчуванням як символ-роздільник лексем використовується пробіл). Далі необхідно викликати функцію `GetNextToken` (отримати наступну лексему) до тих пір, поки `HasMoreTokens` (чи є лексеми) не поверне `false`. Наприклад:

```
wxStringTokenizer tkz(wxT("first: second: third: fourth"), wxT(":"));
while (tkz.HasMoreTokens() ){
 wxString token = tkz.GetNextToken();
 // обробка лексеми
}
```

За замовчуванням, коли символом-роздільником є пробіл, `wxStringTokenizer` поводиться аналогічно функції `strtok`. Проте клас також поверне порожні лексеми (якщо такі будуть). Ця можливість корисна в розборі строго форматованих даних, коли фіксована кількість полів, але деякі з них можуть бути порожніми, як, наприклад, у разі файлу з комами або символами табуляції в ролі роздільників.

Поведінка `wxStringTokenizer` залежить від останнього параметра конструктора, який може приймати одне з таких значень:

- `wxTOKEN_DEFAULT` – використовується поведінка, описана раніше, а саме аналогічно `wxTOKEN_STRTOK` у разі, коли рядок із роздільниками складається тільки з пропусків, і як із `wxTOKEN_RET_EMPTY` в протилежному разі.
- `wxTOKEN_RET_EMPTY` – у режимі повертаються порожні лексеми всередині рядка. Тому рядок `"a::b:"` розіб'ється на три лексеми – `a`, `""` (порожній рядок) і `b`.
- `wxTOKEN_RET_EMPTY_ALL` – повертаються порожні завершальні лексеми (після останнього символу-роздільника). Отже, `"a::b:"` буде розбитий на чотири лексеми – три такі, як у разі `wxTOKEN_RET_EMPTY`, і порожня лексема в кінці.

– wxTOKEN_RET_DELIMS – роздільник у кінці лексеми приєднується до самої лексеми (окрім останньої лексеми, у якої такий символ відсутній). Інакше поведінка схожа на wxTOKEN_RET_EMPTY.

– wxTOKEN_STRTOK – клас поводиться так, як і стандартна функція strtok. Порожні лексеми ніколи не повертаються.

Клас wxStringTokenizer має також дві корисні функції:

– CountTokens повертає кількість лексем, що залишилися в рядку, і 0, коли лексем уже не залишилося.

– GetPosition повертає поточну позицію парсеру в оригінальному рядку.

Клас wxRegEx

Клас wxRegEx можна використати для роботи з регулярними виразами. Він підтримує регулярні вирази для пошуку й заміни в рядках. Клас wxRegEx може використати системну бібліотеку (якщо вона доступна і має підтримку регулярних виразів POSIX, як у випадку сучасних варіантів *Unix*, включно з *Linux* і *Mac OS X*) або вбудовану бібліотеку Генрі Спенсера (Henry Spencer). Регулярні вирази, що відповідають стандарту POSIX, поділяються на дві множини: розширені (*extended*) і прості (*basic*). Вбудована бібліотека також уводить третій режим (ускладнений), який не доступний під час використання системної бібліотеки.

Обробка масивів за допомогою класу wxArray

wxWidgets пропонує структуру для реалізації динамічних масивів на базі класу wxArray, який схожий на стандартні масиви мови *C*, де доступ до елементів здійснюється за постійний час. Проте ці масиви динамічні, а значить можуть самостійно виділяти пам'ять у разі недостатньої поточної ємності для додавання нового елемента. Додавання елемента зазвичай здійснюється за константний час, але для цього масиву доводиться частину пам'яті завжди тримати в резерві. wxArray здійснює перевірку на вихід за межі, що в разі помилки призводить до виникнення твердження (assert) у відлагоджувальній збірці або повер-нення довільного значення у фінальній збірці (у цьому випадку ваша програма може отримати довільне значення в операціях із масивами).

У *wxWidgets* існує три типи масивів. Усі ці типи є нащадками від класу wxBaseArray, який працює з даними, що не типізуються, а тому не може бути використаний безпосередньо. Макроси WX_DEFINE_ARRAY, WX_DEFINE_SORTED_ARRAY і WX_DEFINE_OBJARRAY використовуються для створення нових класів-нащадків від нього. На такі класи

зазвичай посилаються за їхніми іменами `wxArray`, `wxSortedArray` і `wxObjArray`, але важливо розуміти, що класів із такими іменами не існує.

`wxArray` підходить для зберігання простих типів і вказівок, які не вважаються повноцінними об'єктами. Якщо в масиві зберігаються вказівки на об'єкти, то вони автоматично не знищуються під час видалення вказівки з масиву. Крім того, усі функції масиву оголошені як підстановлювані (*inline*), тому без наслідків (як у швидкості виконання, так і в розмірі виконуваного файлу) можна оголошувати дуже багато різних типів для масивів. У класу є одне серйозне обмеження: він може використовуватися тільки для зберігання простих типів (`bool`, `char`, `short`, `int`, `long` і їх беззнакових варіантів) або вказівок на будь-який тип. Дані типу `float` або `double` не можуть зберігається в `wxArray`.

`wxSortedArray` – це варіант класу `wxArray`, який необхідно використати, коли операція пошуку по масиву виконується дуже часто. Під час використання `wxSortedArray` необхідно додатково оголосити функцію для порівняння двох елементів масиву. Елементи в масиві сортуються відповідно до цієї функції. Якщо ви шукаєте в масиві в багато разів частіше, ніж додаєте в нього нову інформацію, то `wxSortedArray` може дати величезний приріст у швидкості, порівняно з `wxArray`. Помітимо, що на типи, що зберігаються в `wxSortedArray`, діють ті ж обмеження, що й на типи в класі `wxArray`. Отже, у цих масивах можна зберігати тільки прості типи даних або вказівки.

Клас `wxObjArray` трактує елементи масиву як об'єкти. Він може видаляти ці елементи, коли вони видаляються з масиву, коректно викликаючи при цьому деструктори. Він також копіює об'єкти, використовуючи конструктор копіювання самого об'єкта. Визначення `wxObjArray` складається з двох частин. По-перше, потрібно оголосити новий клас `wxObjArray`, використовуючи макрос `WX_DECLARE_OBJARRAY`. По-друге, треба підключити файл `<wx/arrimpl.cpp>`, який визначає реалізацію шаблону, і визначити реалізацію вашого класу за допомогою макросу `WX_DEFINE_OBJARRAY` у точці, де вже повністю визначений клас елементів, що містяться в масиві.

Клас масиву рядків `wxArrayString`

`wxArrayString` – це ефективний контейнер, призначений для зберігання об'єктів типу `wxString`, який має ті ж можливості, що й

wxArray. Він також дуже маленький і займає в пам'яті не набагато більше місця, ніж стандартний масив мови *C* типу wxString[] (wxArrayString для цього використовує інформацію про внутрішню структуру класу wxString). Усі методи, доступні в класі wxArray, також доступні й у wxArrayString.

Цей клас використовується аналогічно іншим динамічним масивам, з тією лише різницею, що не треба його оголошувати за допомогою WX_DEFINE_ARRAY, і тип wxArrayString можна використати безпосередньо. Коли рядок додається або вставляється в масив, створюється копія рядка, а початковий рядок може бути безпечно видалений. Отже, не треба турбуватися про управління пам'яттю під час використання цього класу, оскільки він завжди звільняє пам'ять, яку використовує.

Посилання, яке повертається методами Item, Last або operator[], не є константною, а тому можна відразу модифікувати об'єкт, наприклад:

```
array.Last().MakeUpper();
```

Існує також сортований варіант wxArrayString, який називається wxSortedArrayString. Він підтримує ті самі методи, але містить усі текстові рядки, відсортовані в алфавітному порядку. wxSortedArrayString використовує бінарний пошук під час виконання методу Index, що дуже ефективно, якщо рядки рідко додаються в масив і частіше здійснюється по них пошук. Методи Insert і Sort не повинні викликатися в wxSortedArrayString, оскільки це може порушити порядок елементів у масиві.

Створення та знищення масивів. Управління пам'яттю

Класи масивів є повноцінними об'єктами *C++* і реалізують конструктор копіювання та оператор присвоювання. Копіювання wxArray копіює весь його вміст, а копіювання wxObjArray викликає конструктор копіювання для всіх його елементів. Проте для підвищення ефективності роботи з пам'яттю жоден із цих класів не має віртуального деструктора. Це не дуже важливо для wxArray, у якого тривіальний деструктор, але це означає, що потрібно уникати видалення wxObjArray через вказівку на wxBaseArray і не наслідувати власний клас від цих класів-масивів.

Автоматичне управління пам'яттю в масиві дуже просте: масив починає роботу з попереднього виділення деякої кількості пам'яті, яка задається константою WX_ARRAY_DEFAULT_INITIAL_SIZE; коли під час додавання елементів необхідна пам'ять перевищить виділене значення, масив робить перерозподіл, збільшуючи ємність масиву

на 50 % від його поточного об'єму, але не більше ARRAY_MAXSIZE_INCREMENT. Метод Shrink звільняє будь-яку незайняту пам'ять. Метод Alloc може бути дуже корисний, якщо точно відома кількість елементів, розміщуваних у масиві. Використовуючи цей метод, можна резервувати пам'ять для заданої кількості елементів, що дозволить уникнути занадто частого перерозподілу пам'яті.

Класи списку wxList

Клас wxList представляє двозв'язний список, який може зберігати дані довільного типу. *wxWidgets* вимагає явного визначення нового типу списку для кожного окремого типу даних, що дозволяє бібліотеці проводити строгу перевірку типів для значень, що зберігаються в списку. Клас wxList також дозволяє додатково визначити тип значень для ключа, за допомогою якого можна здійснити примітивний пошук.

wxList використовує абстрактний клас wxNode (вузол списку). Якщо оголошується новий тип списку, то також створюється тип для вузлів, що є нащадком від wxNodeBase і дозволяє здійснити строгу перевірку типу. Найважливіші методи класу – GetNext, GetPrevious і GetData – повертають наступний і попередній вузол, а також дані поточного вузла відповідно.

Помітимо, що видалення зі списку працює не зовсім так, як від нього очікується. За замовчуванням під час видалення вузла не видаляються дані, які в ньому містяться. Використовуючи метод DeleteContents, можна змінити цю поведінку – список видалить дані разом із вузлом. Якщо потрібно повністю очистити список і всі дані в ньому, то викликається метод DeleteContents із параметром true перед викликом Clear.

Клас відображення wxHashMap

wxHashMap – це простий клас, що типізується та є досить ефективним для створення відображень (*hash maps*), інтерфейс якого частково повторює інтерфейс відпо-відного *STL*-контейнера.

Зокрема, він моделює частину поведінки стандартного класу std::map і нестандартного std::hash_map. Використовуючи спеціальні макроси для створення хеш-карт, можна вибирати тип для ключів і значень, включно з int, wxString або void* (для зберігання об'єктів довільних класів).

Існує три макроси для оголошення відображень.

1. Оголошення відображення CLASSNAME типом wxString, що використовується як ключ, і типом VALUE_T для значень, виглядає так:

```
WX_DECLARE_STRING_HASH_MAP(VALUE_T, CLASSNAME);
```

2. Оголошення відображення `CLASSNAME` з ключами `void*` і значеннями типу `VALUE_T`:

```
WX_DECLARE_VOIDPTR_HASH_MAP(VALUE_T, CLASSNAME);
```

3. Оголошення відображення `CLASSNAME` із заданим ключем і значенням у загальному випадку має вигляд:

```
WX_DECLARE_HASH_MAP(KEY_T, VALUE_T, HASH_T,
                    KEY_EQ_T, CLASSNAME);
```

де `HASH_T` і `KEY_EQ_T` – типи для хешування та порівняння ключів.

У *wxWidgets* реалізовано три хешуючі функції:

- `wxInteger` – для цілочисельних типів (`int`, `long`, `short` і їхніх беззнакових версій);
- `wxStringHash` – для рядків (`wxString`, `wxChar*`, `char*`);
- `wxPointerHash` – для вказівок на будь-які типи.

Аналогічно визначено три предикати рівності: `wxIntegerEqual`, `wxStringEqual` і `wxPointerEqual`.

Зберігання й обробка дат і часу за допомогою класу `wxDateTime`

wxWidgets надає досить комплексний клас `wxDateTime` для зберігання дати й часу з множиною доступних операцій, а саме: форматування, часові пояси, різні арифметичні операції тощо. Статичні функції дають інформацію про поточну дату й час, а також методи отримання інформації про високосний рік. Допоміжні класи `wxTimeSpan` і `wxDateSpan` дають можливість легко змінювати існуючі об'єкти `wxDateTime`.

Клас `wxDateTime` містить надто багато методів, щоб їх усі розглядати тут. Повну документацію щодо API можна знайти в документації *wxWidgets*. Тут дається опис тільки найчастіше використовуваних методів та операцій.

Хоча під час роботи клас часу припускає, що час відлічується по Грінвічу (*Greenwich Mean Time – GMT*), про це не потрібно турбуватися, оскільки зазвичай доводиться працювати в одному часовому поясі (локальному часі). Отже, усі конструктори й модифікатори `wxDateTime`, що становлять дату або час із компонентів (наприклад, із годин, хвилин і секунд), припускають, що ці значення даються для локального часу. Усі методи, що повертають компоненти дати або часу (місяць, день, годину, хвилину, секунду тощо) також дають правильні значення для локального часового поясу.

Об'єкт `wxDateTime` може бути сконструйований зі значення типу `time_T` (час Unix), інформації про дату, інформації про час або повну інформацію про дату й час. Для кожного конструктора існує відповідний

метод `Set`, який змінює існуючий об'єкт так, щоб він мав певний час або дату. Аналогічно існують модифікатори для індивідуальних компонентів, зокрема `SetMonth` або `SetHour`, які змінюють тільки одну компоненту дати або часу.

Нижче наведено перелік конструкторів `wxDateTime`.

1. `wxDateTime(time_T)` – створює об'єкт із датою та часом відповідно до значення часу *Unix*.

2. `wxDateTime(const struct tm&)` створює об'єкт, використовуючи дані зі стандартної структури `tm` мови *C*.

3. `wxDateTime(wxDateTime_T hour, wxDateTime_T minute=0, wxDateTime_T second=0, wxDateTime_T millisec=0)` створює об'єкт, керуючись заданою інформацією про час.

4. `wxDateTime(wxDateTime_T day, Month month=Inv_Month, int year=Inv_Year, wxDateTime_T hour=0, wxDateTime_T minute=0, wxDateTime_T second=0, wxDateTime_T millisec=0)` створює об'єкт, керуючись заданою інформацією про час і дату.

Аксесори для `wxDateTime` зазвичай мають зрозумілі імена: `GetYear`, `GetMonth`, `GetDay`, `GetWeekDay`, `GetHour`, `GetMinute`, `GetSecond`, `GetMillisecond`, `GetDayOfYear`, `GetWeekOfYear`, `GetWeekOfMonth` і `GetYearDay`.

Клас `wxDateTime` також має такі корисні методи:

- `GetTicks` повертає дату й час у форматі часу *Unix* (кількість секунд, що пройшли з опівночі 1 січня 1970 року);
- `IsValid` вказує, чи знаходиться в об'єкті допустима дата (можна сконструювати об'єкт, який містить неприпустиму дату);
- `wxDateTime::Now` створює об'єкт `wxDateTime` з поточною датою з точністю до секунди;
- `wxDateTime::UNow` створює об'єкт `wxDateTime` з поточною датою з точністю до мілісекунди.

Два об'єкти `wxDateTime` можна легко порівняти, використовуючи одну з багатьох функцій порівняння. Усі ці методи повертають `true` або `false`.

Наступні методи порівнюють поточний об'єкт часу з іншим об'єктом класу `wxDateTime`: `IsEqualTo` (рівні), `IsEarlierThan` (раніше ніж), `IsLaterThan` (пізніше ніж), `IsSameDate` (одна й та сама дата) і `IsSameTime` (один і той самий час).

Наступні методи роблять порівняння, використовуючи два інші об'єкти `wxDateTime`: `IsStrictlyBetween` (точно між) і

IsBetween (між). Різниця між цими двома методами в тому, що IsStrictlyBetween повертає false, якщо wxDateTime об'єкт дорівнює одному з крайових значень проміжку, тоді як IsBetween поверне в такому разі true.

Бібліотека *wxWidgets* містить два дуже корисних класи для виконання арифметичних операцій з об'єктами wxDateTime – wxTimeSpan і wxDateSpan.

wxTimeSpan просто містить різницю в мілісекундах і завжди дає швидкий і передбачуваний результат.

З іншого боку, час має й інші важливі одиниці виміру, такі як тижні та місяці. wxDateSpan намагається здійснити арифметичні операції найбільш природно, проте результат таких операцій не завжди добре визначений. Наприклад, складання 31 січня та 1-го місяця дасть результат 28 (чи 29) лютого, який є останнім днем лютого, а не 31 лютого (цей день у лютому не існує). Звичайно, у цьому разі ми отримуємо те, що чекаємо, проте віднімання того ж самого інтервалу з 28 лютого дасть результат 28 січня (а не 31 січня).

Можна виконувати безліч різних операцій із датами, але не всі з них мають сенс. Наприклад, множення дати на число є некоректною операцією, проте множення тимчасового інтервалу на число коректне.

Перетворення дати на текст досить тривіальне. Можна використати форматування дати й часу відповідно до поточної локалі (FormatDate і FormatTime), використати міжнародний стандарт представлення дати, визначений в *ISO 8601* (*FormatISODate* і *FormatISOTime*), або визначити довільний стандарт, використовуючи метод Format.

Створення дати з рядка – значно цікавіше завдання, тому що існує безліч різних форматів. У найпростішому випадку її можна вирахувати за допомогою методу ParseFormat, який може розібрати будь-яку дату в певному форматі.

ParseRfc822Date прочитує дату у форматі з *RFC 822*, який визначає формат дати для електронної пошти в Інтернеті.

Найцікавішими функціями є ParseTime, ParseDate і ParseDateTime, які намагаються рахувати дату і/або час у «довільному» форматі, дозволяючи визначити їх різними шляхами. Ці функції можна використати для розбору призначеного для користувача введення, яке зазвичай не відповідає жодному зумовленому формату. Наприклад, ParseDateTime може розбирати рядки "tomorrow", "March first", "next Sunday".

Практична робота

1. Створимо додаток, який використовує `wxObjArray` для зберігання даних користувацького типу. Завантажимо *Code::Blocks* та створимо новий проект *wxWidgets* із використанням дизайнера *wxSmith*. Назвемо його *wxCustomers*. Усі інші налаштування проекту приймемо такими, як у попередньому розділі. На форму помістимо два текстових поля з мітками *Cust Id* і *Cust Name*, список, кнопки *Add, Remove, Sort* і *Clear* (рис. 4.1).

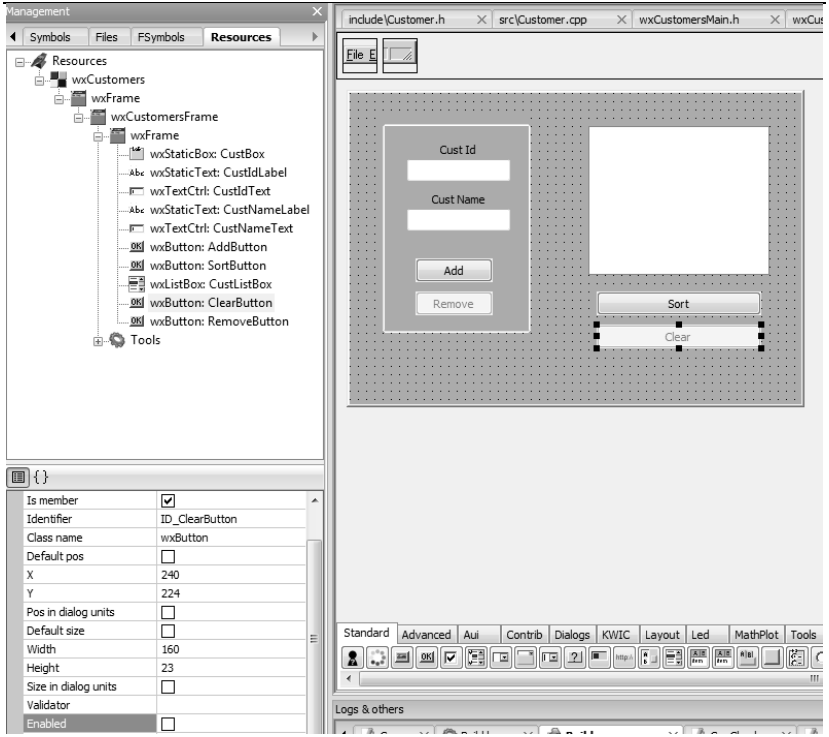

Рис. 4.1. Шаблон користувацького інтерфейсу

Імена змінних відповідних класів та їх ID будуть такими ж, тільки з додаванням суфіксів `Label` або `Text` і префіксів `ID_`. Поля з мітками та кнопку *Add, Remove* розмістимо в межах компонентау `wxGroupBox`, у якого залишимо пустий заголовок.

За допомогою панелі властивостей установимо недоступність кнопок *Remove* та *Clear*, знявши прапорець *Enabled*.

Далі створимо командою *File>>New>>Class…* новий клас `Customer`, у який додамо такі поля: `int CustID`, `wxString CustName` (рис. 4.2).

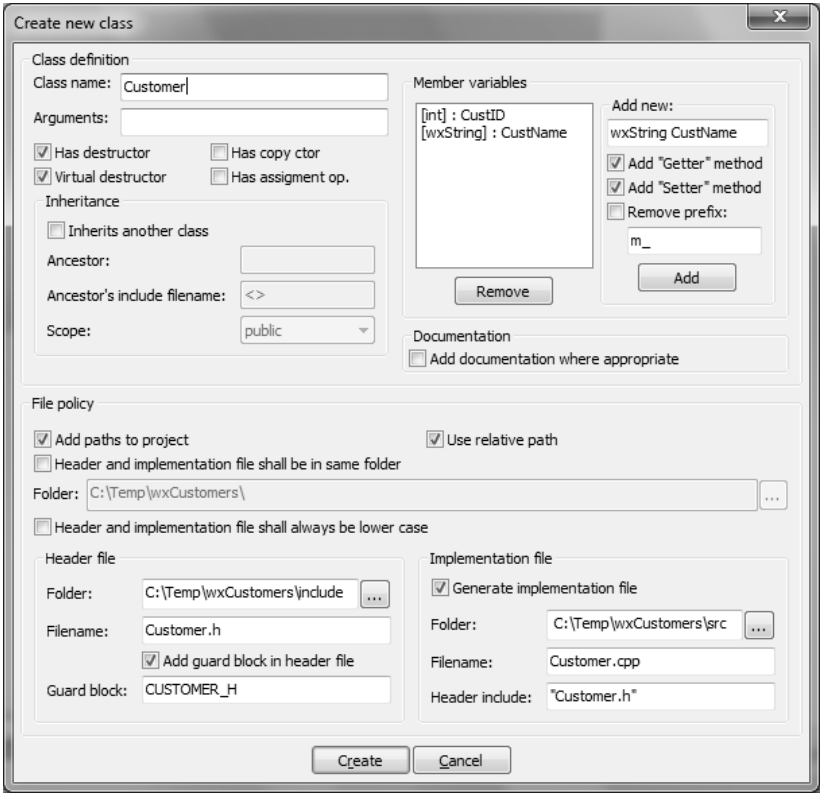

Рис. 4.2. Створення класу `Customer`

Натиснемо кнопку *Create*. Отримаємо заготівлю нашого класу. За оголошенням класу додамо макрос `WX_DECLARE_OBJARRAY` такого змісту:

```
WX_DECLARE_OBJARRAY(Customer, CustomerArray);
```

Цей макрос оголошує й частково реалізує клас `CustomerArray`, який є нащадком від `wxArrayBase`. Для того, щоб макрос був доступним, потрібно підключити заголовочний файл `<wx/dynarray.h>`.

Оголосимо також у файлі заголовка дружню функцію порівняння об'єктів типу `Customer`:

```
friend int arrayCompare(Customer** arg1,
                        Customer** arg2);
```

У лістингу 4.1 наведено код заголовочного файлу *Customer.h*.

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
#include <wx/string.h>
#include <wx/dynarray.h>
class Customer{
 public:
  Customer();
  virtual ~Customer();
  int GetCustID() { return CustID; }
  void SetCustID(int val) { CustID = val; }
  wxString GetCustName() { return CustName; }
  void SetCustName(wxString val) { CustName = val; }
  friend int arrayCompare(Customer** arg1,
                          Customer** arg2);
 protected:
 private:
  int CustID;
  wxString CustName;
};
WX_DECLARE_OBJARRAY(Customer, CustomerArray);
#endif // CUSTOMER_H
```

Лістинг 4.1. Заголовочний файл *Customer.h*

У файлі реалізації підключимо файл wx/arrimpl.cpp та додамо макрос `WX_DEFINE_OBJARRAY(CustomerArray);` а також реалізацію функції порівняння.

Код файлу реалізації наведено в лістингу 4.2.

```
#include "Customer.h"
#include <wx/arrimpl.cpp>
WX_DEFINE_OBJARRAY(CustomerArray);

int arrayCompare(Customer** arg1,Customer** arg2){
  return ((*arg1)->CustID<(*arg2)->CustID);
}
Customer::Customer(){}
Customer::~Customer(){}
```

Лістинг 4.2. Файл реалізації *Customer.cpp*

У файлі заголовка форми помістимо приватне поле `CustomerArray* CustArray`, а у файлі реалізаціїї додамо виділення пам'яті під нього в конструкторі та її звільнення в деструкторі.

Двічі клікнемо на кнопках та згенеруємо обробники подій для них, у які помістимо код, який додає, видаляє елемент у масив, сортує його та очищує. Код заголовочного файлу форми наведено в лістингу 4.3, а код файлу реалізації – у лістингу 4.4.

```
//wxCustomersMain.h
#ifndef WXCUSTOMERSMAIN_H
#define WXCUSTOMERSMAIN_H
//(*Headers(wxCustomersFrame)
#include <wx/stattext.h>
#include <wx/menu.h>
#include <wx/textctrl.h>
#include <wx/listbox.h>
#include <wx/statbox.h>
#include <wx/button.h>
#include <wx/frame.h>
#include <wx/statusbr.h>
//*)
#include "Customer.h"

class wxCustomersFrame: public wxFrame{
 public:
  wxCustomersFrame(wxWindow* parent,wxWindowID id = -1);
  virtual ~wxCustomersFrame();
 private:
 //(*Handlers(wxCustomersFrame)
  void OnQuit(wxCommandEvent& event);
  void OnAbout(wxCommandEvent& event);
  void OnAddButtonClick(wxCommandEvent& event);
  void OnRemoveButtonClick(wxCommandEvent& event);
  void OnSortButtonClick(wxCommandEvent& event);
  void OnClearButtonClick(wxCommandEvent& event);
```

```
    //*)
    //(*Identifiers(wxCustomersFrame)
     static const long ID_CustBox;
     static const long ID_CustIdLabel;
     static const long ID_CustIdText;
     static const long ID_CustNameLabel;
     static const long ID_CustNameText;
     static const long ID_AddButton;
     static const long ID_SortButton;
     static const long ID_CustListBox;
     static const long ID_ClearButton;
     static const long ID_RemoveButton;
     static const long idMenuQuit;
     static const long idMenuAbout;
     static const long ID_STATUSBAR1;
    //*)
    //(*Declarations(wxCustomersFrame)
     wxTextCtrl* CustIdText;
     wxTextCtrl* CustNameText;
     wxStaticText* CustNameLabel;
     wxStaticText* CustIdLabel;
     wxButton* SortButton;
     wxStatusBar* StatusBar1;
     wxButton* ClearButton;
     wxListBox* CustListBox;
     wxButton* AddButton;
     wxStaticBox* CustBox;
     wxButton* RemoveButton;
    //*)
     CustomerArray* CustArray;
     DECLARE_EVENT_TABLE()
};
#endif // WXCUSTOMERSMAIN_H
```

Лістинг 4.3. Код заголовка форми

```
//wxCustomersMain.cpp
#include "wxCustomersMain.h"
#include <wx/msgdlg.h>
//(*InternalHeaders(wxCustomersFrame)
#include <wx/intl.h>
#include <wx/string.h>
//*)
//helper functions
enum wxbuildinfoformat {
 short_f, long_f };
```

```
   wxString wxbuildinfo(wxbuildinfoformat format){
    wxString wxbuild(wxVERSION_STRING);
    if (format == long_f ){
   #if defined(__WXMSW__)
     wxbuild << _T("-Windows");
   #elif defined(__UNIX__)
     wxbuild << _T("-Linux");
   #endif
   #if wxUSE_UNICODE
     wxbuild << _T("-Unicode build");
   #else
     wxbuild << _T("-ANSI build");
   #endif // wxUSE_UNICODE
    }

    return wxbuild;
   }

   //(*IdInit(wxCustomersFrame)
   const long wxCustomersFrame::ID_CustBox = wxNewId();
   const long wxCustomersFrame::ID_CustIdLabel = wxNewId();
   const long wxCustomersFrame::ID_CustIdText = wxNewId();
   const    long    wxCustomersFrame::ID_CustNameLabel    =
wxNewId();
   const long wxCustomersFrame::ID_CustNameText = wxNewId();
   const long wxCustomersFrame::ID_AddButton = wxNewId();
   const long wxCustomersFrame::ID_SortButton = wxNewId();
   const long wxCustomersFrame::ID_CustListBox = wxNewId();
   const long wxCustomersFrame::ID_ClearButton = wxNewId();
   const long wxCustomersFrame::ID_RemoveButton = wxNewId();
   const long wxCustomersFrame::idMenuQuit = wxNewId();
   const long wxCustomersFrame::idMenuAbout = wxNewId();
   const long wxCustomersFrame::ID_STATUSBAR1 = wxNewId();
   //*)

   BEGIN_EVENT_TABLE(wxCustomersFrame,wxFrame)
    //(*EventTable(wxCustomersFrame)
    //*)
   END_EVENT_TABLE()
   wxCustomersFrame::wxCustomersFrame(wxWindow* parent,
                                      wxWindowID id){
    //(*Initialize(wxCustomersFrame)
    wxMenuItem* MenuItem2;
    wxMenuItem* MenuItem1;
    wxMenu* Menu1;
```

```
wxMenuBar* MenuBar1;
wxMenu* Menu2;

Create(parent, wxID_ANY, wxEmptyString,
       wxDefaultPosition, wxDefaultSize,
       wxDEFAULT_FRAME_STYLE, wxT("wxID_ANY"));
SetClientSize(wxSize(439,301));
CustBox = new wxStaticBox(this, ID_CustBox,
             wxEmptyString, wxPoint(32,24),
             wxSize(144,208), 0, wxT("ID_CustBox"));
CustIdLabel = new wxStaticText(this, ID_CustIdLabel,
            wxT("Cust Id"), wxPoint(88,48),
            wxDefaultSize, 0, wxT("ID_CustIdLabel"));
CustIdText = new wxTextCtrl(this, ID_CustIdText,
     wxEmptyString, wxPoint(56,64), wxDefaultSize, 0,
     wxDefaultValidator, wxT("ID_CustIdText"));
CustNameLabel = new wxStaticText(this,
  ID_CustNameLabel, wxT("Cust Name"), wxPoint(80,96),
  wxDefaultSize, 0, wxT("ID_CustNameLabel"));
CustNameText = new wxTextCtrl(this, ID_CustNameText,
    wxEmptyString, wxPoint(56,112), wxDefaultSize, 0,
    wxDefaultValidator, _T("ID_CustNameText"));
AddButton = new wxButton(this, ID_AddButton,
       wxT("Add"), wxPoint(64,160), wxDefaultSize, 0,
       wxDefaultValidator, _T("ID_AddButton"));
SortButton = new wxButton(this, ID_SortButton,
    wxT("Sort"), wxPoint(240,192), wxSize(160,23), 0,
    wxDefaultValidator, _T("ID_SortButton"));
CustListBox = new wxListBox(this, ID_CustListBox,
           wxPoint(232,32), wxSize(176,144), 0, 0, 0,
           wxDefaultValidator, wxT("ID_CustListBox"));
ClearButton = new wxButton(this, ID_ClearButton,
   wxT("Clear"), wxPoint(240,224), wxSize(160,23), 0,
   wxDefaultValidator, wxT("ID_ClearButton"));
ClearButton->Disable();
RemoveButton = new wxButton(this, ID_RemoveButton,
    wxT("Remove"), wxPoint(64,192), wxDefaultSize, 0,
    wxDefaultValidator, wxT("ID_RemoveButton"));
RemoveButton->Disable();
MenuBar1 = new wxMenuBar();
Menu1 = new wxMenu();
MenuItem1 = new wxMenuItem(Menu1,
         idMenuQuit, wxT("Quit\tAlt-F4"),
         wxT("Quit the application"), wxITEM_NORMAL);
Menu1->Append(MenuItem1);
MenuBar1->Append(Menu1, wxT("&File"));
```

```
 Menu2 = new wxMenu();
 MenuItem2 = new wxMenuItem(Menu2,
   idMenuAbout, wxT("About\tF1"),
   wxT("Show info about application"), wxITEM_NORMAL);
 Menu2->Append(MenuItem2);
 MenuBar1->Append(Menu2, wxT("Help"));
 SetMenuBar(MenuBar1);
 StatusBar1 = new wxStatusBar(this, ID_STATUSBAR1, 0,
                              wxT("ID_STATUSBAR1"));
 int __wxStatusBarWidths_1[1] = { -1 };
 int __wxStatusBarStyles_1[1] = { wxSB_NORMAL };
 StatusBar1->SetFieldsCount(1,__wxStatusBarWidths_1);
 StatusBar1->SetStatusStyles(1,__wxStatusBarStyles_1);
 SetStatusBar(StatusBar1);

 Connect(ID_AddButton,wxEVT_COMMAND_BUTTON_CLICKED,
(wxObjectEventFunction)&wxCustomersFrame::OnAddButtonClick);

 Connect(ID_SortButton,wxEVT_COMMAND_BUTTON_CLICKED,
(wxObjectEventFunction)&wxCustomersFrame::OnSortButtonClick);

 Connect(ID_ClearButton,wxEVT_COMMAND_BUTTON_CLICKED,
(wxObjectEventFunction)&wxCustomersFrame::OnClearButtonClick);

 Connect(ID_RemoveButton,wxEVT_COMMAND_BUTTON_CLICKED,
(wxObjectEventFunction)&wxCustomersFrame::OnRemoveButtonClick);

 Connect(idMenuQuit,wxEVT_COMMAND_MENU_SELECTED,
    (wxObjectEventFunction)&wxCustomersFrame::OnQuit);

 Connect(idMenuAbout,wxEVT_COMMAND_MENU_SELECTED,
   (wxObjectEventFunction)&wxCustomersFrame::OnAbout);
 //*)
 CustArray = new CustomerArray();
}

wxCustomersFrame::~wxCustomersFrame(){
 if (CustArray){
  CustArray->Clear();
  delete CustArray;
 }
}

void wxCustomersFrame::OnQuit(wxCommandEvent& event)
{ Close();}
```

```
void wxCustomersFrame::OnAbout(wxCommandEvent& event){
 wxString msg = wxbuildinfo(long_f);
 wxMessageBox(msg, wxT("Welcome to..."));
}

void wxCustomersFrame::OnAddButtonClick(
                        wxCommandEvent& event){
 long id=-1;
 if (!CustIdText->GetValue().IsEmpty() &&
     CustIdText->GetValue().ToLong(&id)){
  for (size_t i=0;i<CustArray->GetCount();++i){
     Customer cust = CustArray->Item(i);
   if (id==cust.GetCustID()){
    return;
   }
  }
  Customer* cust=new Customer();
  cust->SetCustID(id);
  cust->SetCustName(CustNameText->GetValue());
  CustArray->Add(cust);

  wxString line = CustIdText->GetValue()+wxT(":")+
                  CustNameText->GetValue();
  CustListBox->AppendAndEnsureVisible(line);
  RemoveButton->Enable();
  ClearButton->Enable();
 }
}

void wxCustomersFrame::OnRemoveButtonClick(
                                  wxCommandEvent& event){
 if (CustArray->GetCount()>0){
  int i = CustListBox->GetSelection();
  if (i>=0 && i<CustArray->GetCount()){
   CustArray->RemoveAt(i);
   CustListBox->Delete(i);
   if (CustArray->GetCount()==0){
    RemoveButton->Disable();
    ClearButton->Disable();
   }
  }
 }
}

void wxCustomersFrame::OnSortButtonClick(
                                  wxCommandEvent& event){
```

```
 if (!CustArray->IsEmpty()){
  CustArray->Sort(arrayCompare);
  CustListBox->Clear();
  for (size_t i=0;i<CustArray->GetCount();++i){
   Customer cust = CustArray->Item(i);
   wxString line = wxT("");
   line<<cust.GetCustID()<<":"<<cust.GetCustName();
   CustListBox->AppendAndEnsureVisible(line);
  }
 }
}

void wxCustomersFrame::OnClearButtonClick(
                                    wxCommandEvent& event){
 CustArray->Clear();
 CustListBox->Clear();
 RemoveButton->Disable();
 ClearButton->Disable();
}
```

Лістинг 4.4. Код реалізації форми

На рис. 4.3 і 4.4 представлено скріншот додатка з даними до та після сортування відповідно.

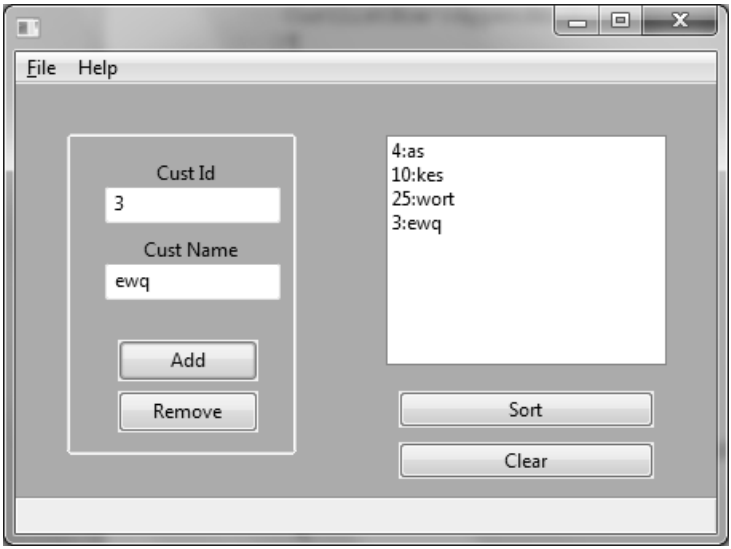

Рис. 4.3. Несортовані дані

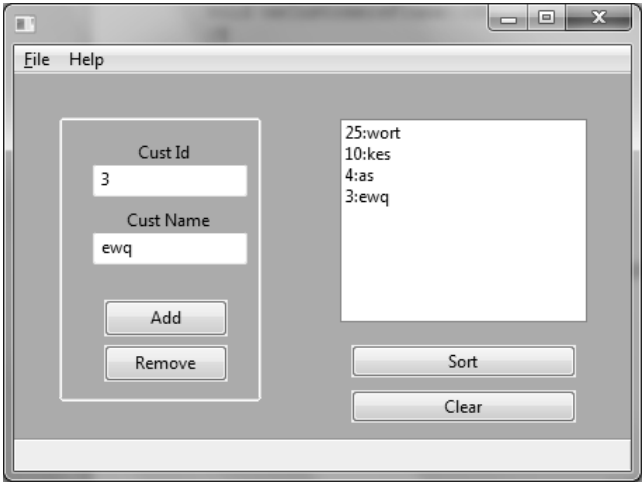

Рис. 4.4. Відсортовані дані

Завдання для самостійної роботи

1. Реалізуйте аналогічну функціональність за допомогою списків та відображень.
2. Додайте можливість сортування за допомогою радіокнопок за зростанням і спаданням.

Тема 5
Допоміжні класи *wxWidgets*. Основи роботи з графікою

Теоретичні відомості

wxWidgets використовує декілька внутрішніх структур даних для передачі в ролі параметрів і повернення результатів із відкритих методів об'єктів. Прикладні програмісти можуть також використати ці допоміжні функції у своїх проектах.

Клас **`wxObject`** є базовим для всіх класів бібліотеки *wxWidgets*. Саме він відповідає за реалізацію отримання інформації про тип часу виконання, підрахунку посилань, оголошення віртуальних деструкторів і додаткові налагоджувальні версії операторів `new` і `delete`.

Клас `wxClassInfo` зберігає метадані про клас і використовується в деяких методах класу `wxObject`, наприклад:

```
MyFrame* frm = wxDynamicCast(FindWindow(ID_MYWINDOW),
                             MyWindow);
```

Метод `IsKindOf` отримує в ролі параметра вказівку на `wxClassInfo` і повертає `true`, якщо об'єкт належить до цього типу, наприклад:

```
bool tmp = obj ->IsKindOf(CLASSINFO(wxFrame));
```

Методу `Ref` передається параметр класу `const wxObject&`, який замінює поточні дані об'єкта на посилання, вказані в ролі параметра. Кількість збережуваних посилань на поточний об'єкт зменшується, можливо звільняються дані, а кількість посилань на об'єкт із параметра збільшується.

`UnRef` зменшує кількість посилань на асоційовані з ним дані й видаляє дані в разі, якщо кількість посилань зменшилася до нуля.

Клас **`wxLongLong`** представляє 64-бітові цілі числа. Коли можливо, то використовується рідний 64-бітовий тип платформи, у решті випадків використовується код, що емулює його. У результаті можна використати цей тип так, як і будь-який інший (вбудований) цілочисельний тип. Помітимо, що число типу `wxLongLong` є числом зі знаком.

Якщо необхідно зберігати беззнакові числа, то використовується для цього тип `wxULongLong`, який має майже схожий API, окрім деяких логічних операторів (зокрема, взяття модуля числа). Для цих

класів визначено звичайні математичні операції, а також декілька додаткових методів:

`Abs` повертає абсолютне значення `wxLongLong`. Якщо цей метод викликати в константного об'єкта, то повернеться копія числа. Для неконстантного об'єкта метод змінить сам об'єкт і поверне посилання на нього.

`ToLong` повертає результат перетворення в тип `long`. Метод викликає виключення в debug-версії, якщо при цьому відбувається втрата точності.

`ToString` повертає рядкове представлення значення числа.

Тип **`wxPoint`** використовується бібліотекою для зберігання пари цілочисельних координат (точки) на екрані або у вікні. Як легко зрозуміти з назви, клас зберігає пару значень `x` і `y`. Ці члени класу є відкритими, а тому з ними можна працювати безпосередньо. Для `wxPoint` реалізовані оператори + і –, які дозволяють додавати або віднімати від поточних координат значення типу `wxPoint` або `wxSize`.

`wxRealPoint` зберігає дійсні (`double`) координати замість цілочисельних, а також реалізує оператори + і –, але тільки для дійсних точок.

Створити об'єкт класу `wxPoint` (або `wxRealPoint`) дуже просто:

```
wxPoint myIntPoint(50, 60);
wxRealPoint myRealPoint(50.14, 16.47);
```

Клас **`wxRect`** використовується для зберігання інформації про прямокутну область і зазвичай використовується бібліотекою *wxWidgets* для малювання або в елементах управління, зокрема `wxDC` або `wxTreeCtrl`. Клас `wxRect` містить поля `x` і `y`, а також `width` (ширина) і `height` (висота), які є відкритими. Прямокутники можна складати й віднімати один з одного. Також клас підтримує декілька інших корисних методів.

- методи `GetRight` і `GetBottom` повертають відповідно координату `x` або `y` правої або нижньої межі прямокутника;
- метод `GetSize` повертає розмір прямокутника (висоту й ширину) у вигляді об'єкта `wxSize`;
- метод `Inflate` збільшує розмір прямокутника на задану величину, яка може бути однаковою для обох напрямів (один параметр) або різною для різних (два параметри);
- метод `Inside` визначає, чи знаходиться задана точка всередині прямокутника. Точка визначається як пара координат або як об'єкт `wxPoint`;

– метод `Intersects` приймає як параметр інший прямокутник `wxRect` і визначає, чи перекриваються вони;

– метод `Offset` переміщує прямокутник на задане зміщення. Зміщення можна задати пару значень або як об'єкт `wxPoint`.

Об'єкт `wxRect` можна створити трьома способами. Наступні три об'єкти визначають один і той же прямокутник:

```
wxRect myRect1(50, 60, 100, 200);
wxRect myRect2(wxPoint(50, 60), wxPoint(150, 260));
wxRect myRect3(wxPoint(50, 60), wxSize(100, 200));
```

Клас **`wxRegion`** представляє просту або складну область на полотні або вікні. Цей клас використовує технологію підрахунку посилань, а тому операції копіювання та присвоєння виконуються для нього дуже швидко. В основному `wxRegion` використовується для визначення областей відсікання або оновлення. Нижче наведено список найчастіше використовуваних методів цього класу.

Метод `Contains` повертає `true`, якщо пара координат (або `wxPoint`), прямокутник або `wxRect` знаходяться всередині цієї області.

Метод `GetBox` повертає об'єкт класу `wxRect`, що представляє прямокутник, який містить область.

Метод `Intersect` повертає `true`, якщо вибраний прямокутник `wxRect` або `wxRegion` перетинає область.

Метод `Offset` переміщує область на певне зміщення. Зміщення визначається у вигляді пари координат `x` і `y`.

Методи `Subtract` (віднімання), `Union` (об'єднання) або `Xor` (виключне або, чи симетрична різниця) змінює регіон різними способами, пропонуючи для цього десяток перевантаженх функцій. У всіх цих методів є версії, що приймають як параметри `wxRegion` або `wxPoint`. Нижче наведено приклади ініціалізації `wxRegion`:

```
wxRegion myRegion1(50, 60, 100, 200);

wxRegion myRegion2(wxPoint(50,60),
                   wxPoint(150, 260));

wxRect myRect(50, 60, 100, 200);
wxRegion myRegion3(myRect);

wxRegion myRegion4(myRegion1);
```

Для перебору всіх прямокутників у заданій області можна використати клас `wxRegionIterator`, наприклад для перемальо-вування «пошкодженої» області екрана в обробнику відповідної події, як показано в прикладі нижче:

```
// Викликається коли вікно необхідно перемальовувати
void MyWindow::OnPaint(wxPaintEvent& event){
 wxPaintDC dc(this);
 wxRegionIterator upd(GetUpdateRegion());
 while (upd){
  wxRect rect(upd.GetRect());
  //перемальовували прямокутник
  ... тут йде код ...
  upd++;
 }
}
```

Клас **wxSize** використовується всередині *wxWidgets* для зберігання цілочисельних розмірів вікна, елементів управління, об'єктів на полотні тощо. Цей об'єкт також часто повертається методами, які повертають інформацію про розмір. Він містить такі стандартні методи:

- `GetHeight` і `GetWidth` – повертають висоту й ширину;
- `SetHeight` і `SetWidth` – приймають цілочисельний параметр, що визначає нове значення для висоти або ширини;
- `Set` установлює нові параметри для розміру.

Об'єкт дуже легко створити, вказавши висоту й ширину:

```
wxSize mySize(100, 200);
```

Клас **wxVariant** є контейнером для будь-якого типу. Значення цього типу може змінюватися в процесі виконання програми, можливо, разом із типом. Цей клас може застосовуватися для зменшення розміру коду в деяких задачах, наприклад у редакторі різних типів даних або реалізації протоколу віддаленого виклику процедур.

`wxVariant` може зберігати значення типу `bool`, `char`, `double`, `long`, `wxString`, `wxArrayString`, `wxList`, `wxDateTime`, вказівку на `void` і масив `wxVariant`. Проте будь-який додаток може розширити можливості `wxVariant`. Для цього необхідно створити нащадка від `wxVariantData` і використати варіант конструктора класу `wxVariant` із параметром `wxVariantData` або оператор присвоювання, щоб присвоїти класу `wxVariant` значення вашого типу. Справжні значення для визначених користувачем типів необхідно отримувати через спеціальні методи об'єкта `wxVariantData`. Це відрізняється від використання стандартних типів, для яких можна викликати вбудовані функції, зокрема `GetLong`.

Потрібно також пам'ятати, що не всі типи даних може бути автоматично перетворено на інші типи. Наприклад, відсутнє перетворення з логічного типу на об'єкт `wxDateTime` і з цілочисельного – на `wxArrayString`.

У будь-який момент можна дізнатися, який поточний тип зберігається в об'єкті класу `wxVariant`, використовуючи для цього метод `GetType`. Нижче наведено приклад використання `wxVariant`:

```
wxVariant Var;
//Зберігаємо wxDateTime і отримуємо wxString
Var=wxDateTime::Now();
wxString DateAsString=Var.GetString();
//Зберігаємо wxString, а отримуємо дійсне число
Var=wxT("10.25");
double StringAsDouble = Var.GetDouble();
// Тип має бути "string" (рядок)
wxString Type=Var.GetType();
//Не можна перетворити рядоку число,
//тому метод поверне 0
char c=Var.GetChar();
```

Практична робота

Створимо простий графічний редактор, у якому будемо малювати найпростіші геометричні фігури – лінію, еліпс, прямокутник. Причому в разі, коли натиснуто клавішу *Shift*, повинні малюватися коло та квадрат.

Завантажимо Code::Blocks, створимо звичайним чином проект *wxWidgets* і назвемо його *wxRects*. У дизайнері додамо лише один пункт меню *Paint* з опціями малювання – *Line* (лінія), *Rectangle* (прямокутник), *Ellipse* (еліпс). Дамо цим пунктам меню відповідні назви, ідентифікатори *idLineMenuOpt*, *idRectMenuOpt*, *idEllipseMenuOpt* та назви полів класу форми *LineMenuOpt*, *RectMenuOpt*, *EllipseMenuOpt* відповідно (рис. 5.1). Тип усіх цих елементів меню має бути *Radio*.

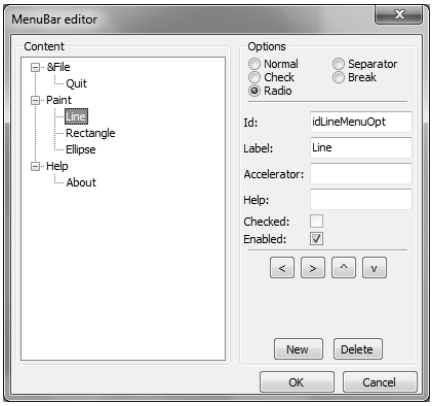

Рис. 5.1. Додавання опцій малювання в дизайнері

У заголовочному файлі форми додамо перелічення `ShapeType` виду

```
enum ShapeType {SHAPE_LINE, SHAPE_RECT,
                SHAPE_ELLIPSE, SHAPE_POLYLINE};
```

а також змінну цього типу, яка буде вказувати на поточний тип фігури, яку потрібно малювати:

```
ShapeType currentShape=SHAPE_LINE;
```

Додамо також такі змінні:

```
wxPoint* startDrawPoint=NULL;
wxPoint* stopDrawPoint=NULL;
bool isDrawing=false;
```

які будуть вказувати відповідно на початкову та кінцеву точки фігури і чи натиснута ліва кнопка миші.

Додамо також користувацьку функцію малювання фігури такого змісту (лістинг 5.1):

```
void wxRectsFrame::DrawUserShape(ShapeType shape,
                                 wxClientDC& dc,
                                 wxPen& pen,
                                 wxPoint& startPoint,
                                 wxPoint& stopPoint,
                                 bool sameDims){
 dc.SetPen(pen);
 wxRect rect(startPoint,stopPoint);
 if (sameDims){
  wxSize sz=rect.GetSize();
  if (sz.x<sz.y){
   sz.Set(sz.x,sz.x);
  }
  else{
   sz.Set(sz.y,sz.y);
  }
  rect.SetSize(sz);
 }
 switch(shape){
  case SHAPE_LINE:
   dc.DrawLine(startPoint,stopPoint);
   break;
  case SHAPE_RECT:
   dc.DrawRectangle(rect);
   break;
  case SHAPE_ELLIPSE:
   dc.DrawEllipse(rect);
   break;
 }
}
```

Лістинг 5.1. Функція малювання фігур

Врахуємо, що під час перетягування миші фігура повинна перемальовуватися, а попередній її стан повинен зникати. Цього можна досягти в найпростішому випадку, малюючи фігуру двічі: один раз кольором фону, а другий – заданим кольором.

У дизайнері форми додамо обробники таких подій: натискання та відпускання лівої кнопки миші `EVT_LEFT_DOWN` і `EVT_LEFT_UP`, а також події переміщення курсору миші `EVT_MOTION`. У лістингу 5.2 наведено код обробників цих подій:

```
void wxRectsFrame::OnMouseMove(wxMouseEvent& event){
 wxString res=wxT("");
 if (isDrawing){
  wxClientDC dc(this);
  wxBrush backBrush=dc.GetBackground();
  wxBrush brush=dc.GetBrush();
  brush.SetColour(backBrush.GetColour());
  dc.SetBrush(brush);
  wxPen* pen = new wxPen(backBrush.GetColour(),1);

  DrawUserShape(currentShape,dc,*pen, *startDrawPoint,
                *stopDrawPoint, event.ShiftDown());
  delete stopDrawPoint;
  stopDrawPoint=new wxPoint(event.GetPosition());
  delete pen;
  pen = new wxPen(*wxBLACK,1);
  DrawUserShape(currentShape,dc,*pen, *startDrawPoint,
                *stopDrawPoint, event.ShiftDown());
  delete pen;
 }
 if (stopDrawPoint){
  res << "Current position (X: " << stopDrawPoint->x;
  res << "; Y: " << stopDrawPoint->y<<")";
 }
 StatusBar1->SetLabel(res);
}

void wxRectsFrame::OnLeftDown(wxMouseEvent& event){
 startDrawPoint = new wxPoint(event.GetPosition());
 stopDrawPoint = new wxPoint(event.GetPosition());
 isDrawing=true;
}

void wxRectsFrame::OnLeftUp(wxMouseEvent& event){
 wxClientDC dc(this);
 wxBrush backBrush=dc.GetBackground();
 wxBrush brush=dc.GetBrush();
 brush.SetColour(backBrush.GetColour());
 dc.SetBrush(brush);
 wxPen* pen = new wxPen(backBrush.GetColour(),1);
 DrawUserShape(currentShape,dc,*pen, *startDrawPoint,
```

```
                *stopDrawPoint, event.ShiftDown());
  delete stopDrawPoint;
  stopDrawPoint=new wxPoint(event.GetPosition());
  delete pen;
  pen = new wxPen(*wxBLACK,1);
  DrawUserShape(currentShape,dc,*pen, *startDrawPoint,
                *stopDrawPoint, event.ShiftDown());
  delete pen;
  delete startDrawPoint;
  delete stopDrawPoint;
  startDrawPoint=stopDrawPoint=NULL;
  isDrawing=false;
 }
```

Лістинг 5.2. Оброка подій, які виникають під час малювання фігури

І наостанок додаємо обробники подій натискання на пункти меню для малювання відповідних фігур (лістинг 5.3):

```
void wxRectsFrame::OnLineMenuOptSelected(
                                  wxCommandEvent& event){
 currentShape = SHAPE_LINE;
}

void wxRectsFrame::OnRectMenuOptSelected(
                     wxCommandEvent& event){
 currentShape = SHAPE_RECT;
}

void wxRectsFrame::OnEllipseMenuOptSelected(
                                  wxCommandEvent& event){
 currentShape = SHAPE_ELLIPSE;
}
```

Лістинг 5.3. Вибір типу фігури

На рис. 5.2, 5.3 представлено скріншоти цієї програми:

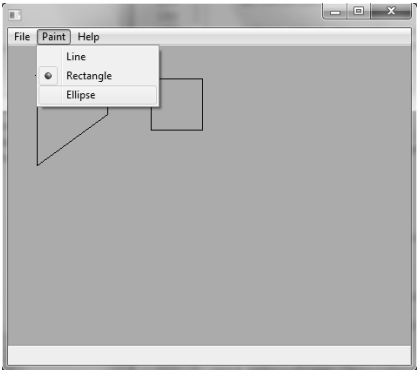

Рис. 5.2. Головне вікно прорами з лінією та квадратом

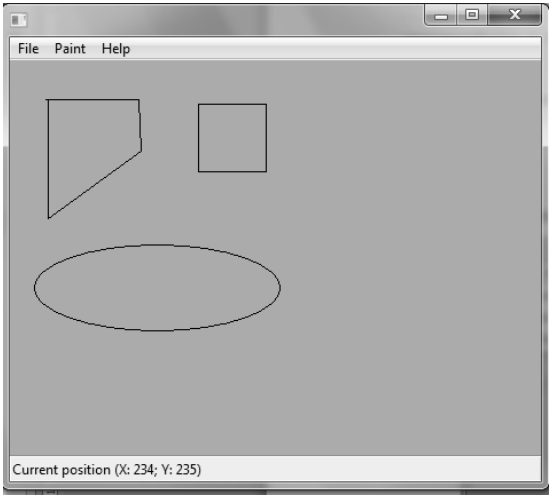

Рис. 5.3. Головне вікно програми з еліпсом

Завдання для самостійної роботи

1. Додайте можливість малювання поліліній.
2. Додайте функціональність, яка б дозволяла отримувати інформацію про розміри та площі мальованих фігур.

ЛІТЕРАТУРА

1. Smart J. Cross-Platform GUI Programming with wxWidgets [Text] / Julian Smart, Kevin Hock. – Prentice Hall, 2007. – 744 p.
2. [Електронний ресурс]. – Режим доступу : http://docs.wxwidgets.org.
3. [Електронний ресурс]. – Режим доступу : http://wiki.wxwidgets.org/Main_Page.

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

Навчальне видання

Пузирьов

Сергій Володимирович

ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ МЕРЕЖ ІЗ ВИКОРИСТАННЯМ БІБЛІОТЕКИ *wxWidgets*

Методичні рекомендації

Випуск 240

Редактор, технічний редактор *С. Сидоренко.*
Комп'ютерна верстка *О. Гончарова.*
Друк, фальцювально-палітурні роботи *С. Волинець.*

Підп. до друку 07.12.2015.
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Verdana». Друк ризограф.
Ум. друк. арк. 4,41. Обл.-вид. арк. 2,25.
Тираж 40 пр. Зам. № 4766.

Видавець і виготовлювач: ЧДУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50-03-32, 8 (0512) 76-55-81, e-mail: vrector@chdu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 3460 від 10.04.2009