7. Własny edytor C++, czyli bardziej zaawansowane kontrolki i techniki wxWidgets

Poznałeś już techniki i kontrolki GUI, a także podstawowe zasady rządzące biblioteką wxWidgets, które pozwalają na tworzenie prostych programów obsługujących elementarną interakcję z użytkownikiem. Teraz zapoznam Cię z kolejnymi tajnikami wxWidgets, które nie tylko pozwolą Ci powtórzyć ten materiał, ale także poszerzą Twój warsztat i pozwolą na realizację bardziej złożonych projektów.

Rozpoczęcie nauki programowania od próby tworzenia aplikacji w stylu notatnika jest bardzo częstą praktyką na wszelkiego rodzaju kursach i w poradnikach. Choć nie jest to początek, a Twoja wiedza ma solidne podstawy, w tym rozdziale zajmiemy się stworzeniem własnego notatnika, będącego formą nieco bardziej rozbudowanego edytora C++, który będziesz mógł później rozwinąć według własnej fantazji. Myślę, że będzie on bardzo atrakcyjny do wykonania – zwłaszcza, że jego funkcjonalność będzie stanowiła zalążek znanej Ci z innych programów tego typu.

Abyś mógł utworzyć własny edytor C++, konieczne będzie poznanie takich zagadnień, jak:

- użycie zaawansowanej kontrolki tekstowej *wxStyledTextCtrl* obsługującej kolorowanie składni oraz numerację linii kodu;
- użycie zaawansowanych kontenerów kontrolek, takich jak *wxSplitterWindow* lub *wxSashLayoutWindow* oraz *wxNotebook*;
- użycie i obsługa zaawansowanych kontrolek, takich jak *wxTreeCtrl* oraz *wxListCtrl*;
- użycie wybranych standardowych okien dialogowych wxWidgets, w tym dotyczących pracy z plikami oraz wyszukiwania danych;
- wykorzystanie współpracy wxWidgets i uniwersalnego języka znaczników XML[1] (*Extensible Markup Language*) do efektywnego zarządzania danymi projektu, czyli wykorzystanie możliwości biblioteki składowej *wxXML*;
- użycie prostych mechanizmów drukowania dokumentów za pomocą możliwości składowej biblioteki *wxHTML*;
- użycie podstawowych mechanizmów wxWidgets służących do pracy z plikami i na plikach;
- użycie podstawowych kontrolek GUI, o których nie było dotychczas mowy.

Tradycyjnie już, przy okazji poruszania poszczególnych tematów, spróbujemy zagłębić się nieco w zagadnienia im pokrewne, dzięki czemu szybko bedziesz miał pełny obraz możliwości wxWidgets i zdobędziesz gruntowną wiedzę. Jestem przekonany, że już podczas lektury tego rozdziału zaczniesz tworzyć szybkie eksperymenty programistyczne.

Zagadnienia, które zostały omówione w poprzednich rozdziałach, będą tu traktowane pobieżnie i tylko w ramach skonkretyzowanego powtórzenia kluczowych wiadomości, dlatego przygotuj się na odrobinę samodzielności. Do dzieła...

---

[1] Nie przejmuj się, jeśli nie znasz języka XML. Oprócz tego, że jest on niezwykle użyteczny, w rzeczywistości jest też bardzo łatwy i intuicyjny, o czym postaram się przekonać Cię na kolejnych stronach książki.

7.1. Koncepcja programu

Przedmiotem naszych aktualnych rozważań będzie utworzenie edytora tekstu, który z założenia ma przypominać edytory z zaawansowanych środowisk programistycznych. Nasz edytor umożliwi kolorowanie składni C++ i wxWidgets, a także ułatwi zarządzanie plikami źródłowymi dzięki umieszczeniu ich w uporządkowanej strukturze, którą ogólnie można nazwać *projektem*. To kolejna praktyka, jaka jest Ci z pewności znana z wszelkich narzędzi IDE, z którymi miałeś styczność i za pomocą których tworzysz swoje programy. Wzorem gigantów wśród wszelkiego rodzaju edytorów programistycznych (w tym Visual C++, Code::Blocks) strukturę projektu wyświetlimy w formie drzewa plików projektu, którego układ, wraz z informacją o typach plików, zapiszemy w pliku wykorzystującym język XML.

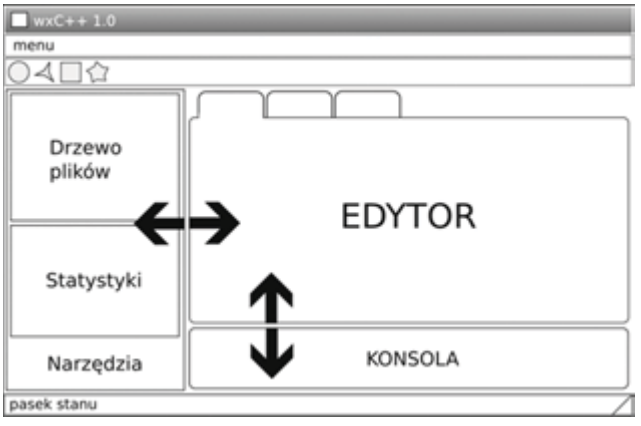
Program będzie zawierał standardowe narzędzia edycyjne umożliwiające wykonywanie wszystkich najważniejszych operacji na tekście, a także szybko dostępne narzędzia pozwalające na natychmiastową zmianę trybu wyświetlania tekstu. Ponadto do naszej aplikacji dodamy zalążek „wyjścia konsoli", do którego będziesz mógł w przyszłości skierować na przykład wyjście kompilatora.

Oprócz tego wyposażymy program w listę statystyk, które będą zawierać bieżące informacje o samym projekcie, jak i o przypisanych do niego plikach źródłowych. Będą to m.in. dane o statusie projektu oraz na temat struktury i zawartości plików, obejmujące na przykład łączną liczbę linii kodu we wszystkich plikach i w pliku aktualnie wyświetlanym. Poszczególne pliki kodu czy tekstu będą wyświetlane w odrębnych, standardowych zakładkach.

Zaimplementujemy również prosty mechanizm drukowania, który umożliwi drukowanie treści bieżąco wyświetlanego dokumentu. Ponadto, aby nieco urozmaicić naszą aplikację, dodamy do niej również prosty kreator klas C++, a także zaimplementujemy prosty mechanizm wyszukiwania i zamiany ciągów znaków w dokumentach projektu.

Graficzny interfejs programu będzie oparty na układzie dwukolumnowym, przy czym wzajemny stosunek szerokości kolumn będzie mógł być sterowany przez użytkownika dzięki uchwyceniu i przeciągnięciu rynny znajdującej się między kolumnami. To samo będzie dotyczyć okna edytora oraz okna wyjścia konsoli, które będą wspólnie zajmować kolumnę prawą. Sam edytor będzie umożliwiał jednoczesną edycję kilku plików, których treść będzie wyświetlana w zakładkach edytora.

Ogólną koncepcję GUI naszego edytora przedstawia rysunek 7.1.



Rysunek 7.1. Koncepcja graficznego interfejsu użytkownika edytora C++, jaki będziemy realizować w niniejszym rozdziale

Nie wyczerpie to oczywiście wszystkich możliwości i opcji, jakie powinna mieć aplikacja tego typu, niemniej jednak będzie już doskonałą podstawą do tego, abyś mógł utworzyć własne kompletne środowisko IDE – rzecz jasna, jak tylko skończysz lekturę książki.