Rozdział 3

Tworzenie aplikacji GUI

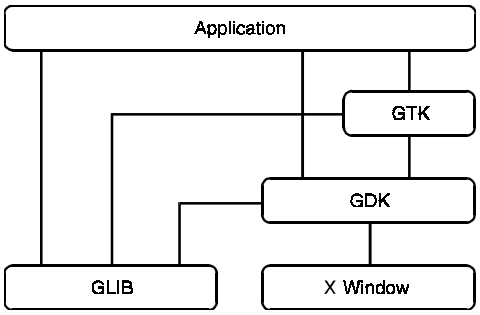
W Linuksie nie ma żadnego pojedynczego standardu tworzenia aplikacji z graficznym interfejsem użytkownika (GUI) - nie licząc biblioteki X Windows (Xlib), która jest bardzo niskopoziomowa i nie nadaje się do szybkiego tworzenia aplikacji. Archiwa w Internecie od dawna były wypełnione pakietami narzędziowymi, które miały ułatwiać pisanie aplikacji, ale żaden z nich nie zdobył wiodącej pozycji, a społeczność Linuksa podzieliła się na frakcje. Sytuacja zaczęła się zmieniać, kiedy grupa TrollTech (www.troll.no) stworzyła bardzo dobry pakiet narzędziowy GUI o nazwie Qt, znakomicie nadający się do tworzenia aplikacji GUI w Linuksie. Jednakże ograniczenia licencyjne zniechęciły wielu programistów do używania tego pakietu. Licencja Qt wymaga, aby stworzone na jego podstawie aplikacje były darmowe, o ile twórca programu nie zakupi komercyjnej wersji biblioteki Qt. Zdaniem autora, tego rodzaju wymaganie stanowi istotne ograniczenie, z powodu którego komercyjne firmy rezygnują z wykorzystania bardzo udanej biblioteki. Niemniej, biblioteka Qt jest bezpłatna w przypadku niekomercyjnych zastosowań.

Linux jest objęty Powszechną Licencją Publiczną GNU (GNU *General Public License*, GPL), która umożliwia każdemu swobodny dostęp do kodu źródłowego, modyfikowanie go i dalsze rozpowszechnianie. Dlatego wielu twórców oprogramowania uważa, że wszystkie pakiety narzędziowe używane do pisania aplikacji dla Linuksa powinny również być objęte licencją GPL. Dzięki takiemu podejściu nie trzeba czekać na poprawienie usterek przez komercyjną firmę, a twórcy oprogramowania nie muszą obawiać się, że zbyt wiele władzy spocznie w jednym ręku. Sytuacja ta doprowadziła do powstania pakietu GIMP Toolkit (GTK+).

GIMP Toolkit (GTK+) jest biblioteką GUI objętą licencją GPL; z tego względu pozostaje bezpłatny zarówno w zastosowaniach komercyjnych, jak i niekomercyjnych. GTK+ jest pakietem międzyplatformowym, pracującym w wielu wersjach UNIX-a oraz w Microsoft Windows. Dzięki wsparciu ze strony dystrybutorów oprogramowania dla Linuksa oraz dużej grupie używających go programistów, GTK+ staje się coraz lepszym pakietem, wspomagającym tworzenie aplikacji GUI w Linuksie.

GTK+ otrzymał swoją nazwę z powodu wykorzystania biblioteki w programie GIMP (*Gnu Image Manipulation Project*). GIMP jest doskonałym edytorem graficznym, dorównującym programom komercyjnym, napisanym przy użyciu GTK+. GIMP przypomina Photoshopa; można zapoznać się z nim pod adresem www.gimp.org.

Biblioteka GTK+, choć napisana w C, jest zorientowana obiektowo i dysponuje powiązaniami, umożliwiającymi wykorzystanie jej z poziomu wielu języków, w tym także C++ (przy pomocy „opakowania" o dość nieoczekiwanej nazwie GTK--). Podobnie jak wiele współczesnych pakietów GUI, GTK+ jest sterowana zdarzeniami. Ekran aplikacji tworzy się przy pomocy *kontrolek* (ang. *widgets*): okien, przycisków, pól tekstowych czy też list kombinowanych, którym przypisuje się *funkcje zwrotne* (ang. *callbacks*), służące do przetwarzania pochodzących od nich sygnałów - zwykle zdarzeń, związanych z klawiaturą lub myszą. Kiedy funkcja zwrotna otrzymuje powiadomienie o sygnale, wówczas aplikacja podejmuje odpowiednie działanie. Koncepcja sygnału jest zbliżona do zdarzenia w Javie albo Windows.



Rysunek 3.1. Warstwy aplikacji GTK+.

GTK+ jest biblioteką kontrolek, opartą na bibliotece GIMP Drawing Kit (GDK), która z kolei stanowi „opakowanie" biblioteki Xlib. GTK+ wywołuje GIMP Drawing Kit podczas każdej operacji, związanej z wyświetlaniem kontrolek. GDK w założeniu stanowi zależny od platformy interfejs API (*Application Programming Interface*), umieszczony ponad macierzystym interfejsem graficznym systemu (Xlib, Win32) i tłumaczący wywołania funkcji graficznych. Ponieważ GDK jest warstwą pomiędzy Xlib i GTK+, przeniesienie GTK+ do innego systemu wymaga napisania nowej biblioteki GDK (patrz rysunek 3.1). Zarówno biblioteka GTK+, jak i GDK opierają swoje działanie na bibliotece GLIB, która dostarcza funkcji obsługujących wiele struktur danych, jak listy, drzewa i łańcuchy, a także

funkcji zarządzających pamięcią i procedur obsługi błędów. Bibliotek GLIB i GDK można używać bezpośrednio, niekoniecznie poprzez GTK+.

Kompilowanie aplikacji GTK+

Aby skompilować aplikację GTK+, należy dołączyć główny plik nagłówkowy GTK+, gtk/gtk.h, do każdej części aplikacji, która używa definicji albo wywołań funkcji GTK+. Oprócz tego niezbędna jest konsolidacja z pewną liczbą bibliotek. Na szczęście twórcy GTK+ ułatwili nam zadanie, pisząc program gtk-config. Można skompilować prosty program GTK+ przy pomocy następującego polecenia:

```
gcc -Wall -g przyklad.c -o przyklad `gtk-config -cflags`
`gtk-config -clibs`
```

Znaki, otaczające napisy `gtk-config -cflags` i `gtk-config -clibs`, muszą być apostrofami skierowanymi do tyłu. W powyższym przykładzie kompilujemy program przyklad.c, a `gtk-config -cflags` powoduje uruchomienie programu gtk-config, który generuje opcje dla kompilatora i umieszcza je w wierszu polecenia. Opcje dla konsolidatora (linkera) generowane są przez `gtk-config -clibs` i również umieszczane w wierszu polecenia. Program gtk-config stanowi część GTK+ i musi znajdować się na ścieżce. Wpisanie gtk-config -cflags albo gtk-config -clibs w wierszu poleceń Linuksa pozwala zobaczyć, jakie dokładnie parametry przekazywane są do kompilatora GNU C (gcc).

Inicjowanie GTK+

W każdym programie GTK+ konieczne jest zainicjowanie biblioteki GTK+, poprzez wywołanie funkcji gtk_init. Argumenty aplikacji (argc, argv) są przekazywane do funkcji gtk_init, która sprawdza, czy są w nich obecne któreś spośród opcji GTK+, używanych głównie podczas „odpluskwiania" programów. Jeśli na liście argumentów zostaną znalezione jakieś opcje GTK+, zostaną one usunięte. Po powrocie z gtk_init aplikacja nie będzie miała do nich dostępu.

```
/* --- Inicjujemy GTK --- */
gtk_init (&argc, &argv);
```

Po zainicjowaniu GTK+ można wywoływać pozostałe funkcje biblioteczne.

Tworzenie okien

Kontrolka w GTK+ jest składnikiem interfejsu GUI. Okna, pola wyboru, przyciski czy pola edycyjne - wszystkie te elementy są kontrolkami. Kontrolki i okna są zawsze definiowane jako wskaźniki do struktury GtkWidget. GtkWidget jest uniwersalnym typem danych, używanym przez wszystkie kontrolki i okna w GTK+.

Po zainicjowaniu biblioteki GTK+, większość aplikacji tworzy główne okno. W GTK+ główne okno nazywane jest zwykle *oknem najwyższego poziomu* (ang. *top-level window*). Okna najwyższego poziomu nie posiadają okien macierzystych, ponieważ nie są zawarte w żadnym oknie. W GTK+ między kontrolkami zachodzą relacje rodzic - potomek, gdzie rodzicem jest pojemnik, a potomkiem kontrolka umieszczona wewnątrz pojemnika. Okna najwyższego poziomu nie posiadają okien macierzystych, ale mogą być rodzicami dla innych kontrolek.

Tworzenie kontrolki w GTK+ przebiega dwuetapowo: najpierw tworzy się kontrolkę, a potem czyni się ją widoczną. Możemy zastosować tę technikę do stworzenia okna najwyższego poziomu.

Aby utworzyć okno najwyższego poziomu, należy wywołać funkcję gtk_window_new z parametrem GTK_WINDOW_TOPLEVEL. Funkcja gtk_window_ new zwraca wskaźnik do GtkWidget. Utworzone okno chwilowo nie jest widoczne; trzeba je wyświetlić przy pomocy funkcji gtk_widget_show.

```
/* --- Tworzymy okno najwyższego poziomu w gtk --- */
/* --- zauważmy, że okno na razie NIE JEST widoczne --- */
okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* --- Teraz uwidaczniamy okno --- */
grk_widget_show (okno);
```

Pętla obsługi zdarzeń w GTK+

Po zainicjowaniu GTK+ i umieszczeniu okien i kontrolek na ekranie, aplikacja przekazuje sterowanie do GTK+, aby można było rozpocząć przetwarzanie zdarzeń (ruchów myszy, naciśnięć klawiszy itp.). Funkcja gtk_main nie powróci, dopóki aplikacja nie wywoła gtk_main_quit. Jeśli jednak gtk_main nie powraca, w jaki sposób aplikacja może wywołać gtk_main_quit? Otóż przed wywołaniem gtk_main należy stworzyć funkcje zwrotne i zarejestrować je w GTK+, dzięki czemu pewne sygnały zwrócą sterowanie do aplikacji.

Program GTK+ może być całkiem niewielki. Poniższy przykład powinien zilustrować przebieg sterowania w programie GTK+. Nie ma w nim żadnych funkcji zwrotnych, więc trzeba będzie zakończyć go ręcznie.

```
#include <gtk/gtk.h>

int main (int argc, char *argv[])
{
    GtkWidget *okno;

    /* --- inicjujemy GTK, przekazując argumenty wiersza polecenia */
    gtk_init (&argc, &argv);

    /* --- tworzymy okno; na razie NIE JEST ono widoczne --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- teraz wyświetlamy okno --- */
    gtk_widget_show (okno);

    /* --- główna pętla zdarzeń w gtk. Nie mamy żadnych procedur
    --- obsługi zdarzeń. Hmmm... --- */
    gtk_main ();

    /* --- zwracamy kod wyjściowy --- */
    return 0;
}
```

Chociaż przykład ten można nazwać programem GTK+, z pewnością nie jest to aplikacja, ponieważ nie wykonuje żadnej przydatnej czynności i nie przerywa pracy, kiedy próbujemy go zakończyć. Zanim zasłuży sobie na miano aplikacji, musimy dodać do niego kilka funkcji zwrotnych,

Typy danych w GTK+

Zanim zaczniemy dodawać do programu procedury obsługi zdarzeń, dzięki którym nabierze on nieco rumieńców, musimy dowiedzieć się kilku rzeczy o typach kontrolek GTK+. Kontrolki w GTK+ zazwyczaj wywodzą się od innych kontrolek. Kontrolka przycisku (GtkButton) wywodzi się od kontrolki pojemnika (GtkContainer), która wywodzi się od kontrolki uniwersalnej (GtkWidget), która wywodzi się od obiektu GTK+ (GtkObject). Wszystkie funkcje, tworzące kontrolki, zwracają wskaźnik do GtkWidget; jest to wskaźnik do kontrolki uniwersalnej, i może wymagać przekształcenia przed użyciem w funkcjach, które operują na konkret-

nych kontrolkach. Powód, dla którego wywodzi się jedną kontrolkę z innej, jest bardzo prosty: jeśli większość funkcji spełnianych przez jedną kontrolkę jest taka sama, jak w przypadku innej, po co tracić czas i wysiłek na pisanie nowej kontrolki? Dużo łatwiej jest wywieść ją z już istniejącej, dodając potrzebne funkcje.

Mechanizm dziedziczenia sugeruje, że podczas pisania GTK+ konieczne było posłużenie się językiem C++. Do napisania biblioteki wybrano jednak z kilku przyczyn język C. C jest podstawowym językiem pisania aplikacji w Linuksie. Jest też bardziej przenośny, niż C++, i został ustandaryzowany wcześniej. Standaryzacja C++ odbywa się dopiero obecnie, i nie wszystkie kompilatory potrafią pracować z tym samym zbiorem kodu. Kiedy rozpoczęto projekt GTK+, standard C++ był jeszcze mglistym marzeniem. Od tego czasu C++ przeszedł długą drogę, ale wciąż występują problemy z pracą różnych kompilatorów na różnych platformach (istnieje kilka projektów opakowania GTK+ biblioteką C++ - więcej szczegółów można znaleźć na stronie WWW GTK+).

Funkcje tworzące przycisk GTK+ zwracają wskaźnik typu GtkWidget, a nie GtkButton. Dzięki temu uniwersalne funkcje (w rodzaju gtk_widget_show) mogą pracować z wszystkimi kontrolkami. Wskaźnik GtkWidget zwrócony przez funkcję tworzącą przycisk może zostać rzutowany na typ GtkButton (przy pomocy makra GTK_BUTTON) w celu użycia go w funkcjach operujących tylko na przyciskach, ponieważ wskazuje on na przycisk. Ten sam GtkWidget można rzutować na typ GtkContainer (przy pomocy makra GTK_CONTAINER) w celu użycia go w funkcjach operujących na pojemnikach, ponieważ przycisk jest pojemnikiem.

Chociaż kontrolkę przycisku można przekazać do jednej z funkcji operujących tylko na przyciskach, to kompilator poskarży się, jeśli przekażemy ją jako kontrolkę uniwersalną. Programowanie w GTK+ wymaga rzutowania kontrolki na właściwy typ przed wywołaniem funkcji. Każda kontrolka posiada makro, przekształcające kontrolkę GtkWidget na konkretny typ kontrolki GTK+. GtkWidget można rzutować na GtkButton przy pomocy makra GTK_BUTTON, ale tylko wtedy, jeśli GtkWidget utworzono przy pomocy funkcji tworzącej przyciski albo kontrolki wywiedzione od przycisku. W innym przypadku kompilator co prawda nie zgłosi zastrzeżeń, ale zrobi to biblioteka GTK+, kiedy program zostanie uruchomiony. Funkcje GTK+ sprawdzają typ przekazywanej im kontrolki, więc przekazanie przycisku do funkcji operującej na pojemnikach jest poprawne, ponieważ przyciski *są* pojemnikami, ale przekazanie pojemnika do funkcji operującej na przyciskach spowoduje błąd, ponieważ pojemniki *nie są* przyciskami. Komunikaty o błędach są wyświetlane na konsoli i stanowią nieocenioną pomoc podczas usuwania usterek z aplikacji.

Sygnały i funkcje zwrotne

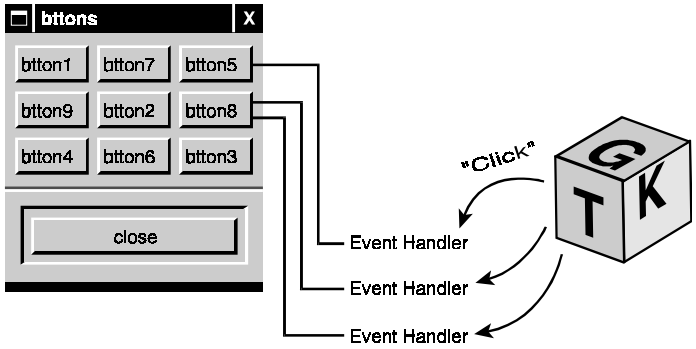
Sygnały są niezbędne w programach opartych na interfejsie GUI, ponieważ program musi mieć możliwość reakcji na działania użytkownika. Jeśli użytkownik przesunie mysz, naciśnie przycisk, wpisze tekst albo zamknie okno, do funkcji zwrotnej w aplikacji zostanie przesłany odpowiedni sygnał. Może to być jeden z sygnałów, które muszą zostać obsłużone przez aplikację. Edytor tekstu może dysponować przyciskiem, który pogrubia czcionkę; jeśli użytkownik naciśnie ten przycisk, musi zostać wywołany kod, który pogrubi czcionkę. Podobnie, jeśli użytkownik zamknie główne okno, potrzebne mogą być pewne dodatkowe czynności (zapisanie plików, posprzątanie po sobie), które należy wykonać przed zamknięciem okna.

Sygnały w aplikacji GTK+ występują niemal bez przerwy, ale większość z nich jest ignorowana. Weźmy jako przykład kontrolkę przycisku. Aplikacje posiadają sygnały specyficzne dla kontrolki przycisku, generowane podczas wciśnięcia przycisku myszy, zwolnienia przycisku myszy, kliknięcia myszą, przesunięcia wskaźnika myszy nad przycisk i opuszczenia przycisku przez wskaźnik myszy. Inne sygnały, zdefiniowane dla kontrolki uniwersalnej, obejmują także sygnał generowany podczas ruchu myszy.

W rzeczywistości jedynym sygnałem, który warto sprawdzać, jest sygnał "clicked". Jest on generowany po kliknięciu przycisku myszą. Aplikacje zazwyczaj ignorują inne zdarzenia związane z przyciskami. Zazwyczaj nie ma sensu, aby aplikacja była informowana o przesunięciu wskaźnika myszy nad przycisk albo zwolnieniu klawisza myszy przez użytkownika. Inne kontrolki przypominają pod tym względem kontrolkę przycisku - tylko nieliczne sygnały są przydatne dla twórcy aplikacji, a wszystkie inne można bezpiecznie zignorować.

Kiedy dany sygnał wymaga obsługi, należy zarejestrować w GTK+ funkcję zwrotną i związać ją z kontrolką. Funkcje zwrotne mogą być zarejestrowane przez wiele kontrolek naraz. Rysunek 3.2 ilustruje związki pomiędzy aplikacją, GTK+ i funkcjami zwrotnymi.

Rysunek 3.2 przedstawia sposób, w jaki obsługiwane są sygnały. Przycisk5 zdefiniował funkcję zwrotną, która będzie wykonywana w razie wystąpienia sygnału. Przycisk8 zdefiniował dwie funkcje zwrotne, które będą czekać na dwa różne sygnały. Kontrolka może odpowiadać na dowolną liczbę sygnałów, ale rozsądnie jest ograniczyć ją do niezbędnego minimum. Kiedy wystąpi sygnał, związany z kontrolką, zostanie wykonana funkcja zwrotna dla tej kontrolki.



Rysunek 3.2. Obsługa sygnałów w GTK+

Dodawanie procedur obsługi sygnału

Dwoma podstawowymi sygnałami w GTK+ są delete_event i destroy. Sygnał delete_event jest wysyłany do okna, które ma zaraz zostać usunięte, natomiast sygnał destroy jest wysyłany do właśnie usuwanego okna. Okno najwyższego poziomu musi posiadać procedurę obsługi dla sygnału delete_event, ponieważ wskazuje on, że użytkownik chce zamknąć aplikację. Dodawanie funkcji zwrotnej dla sygnału delete_event odbywa się w dwóch krokach.

Pierwszym krokiem jest napisanie funkcji zwrotnej, którą w zamieszczonym niżej przykładzie nazwaliśmy usun_aplikacje. Funkcja usun_aplikacje wyświetla komunikat na konsoli i wychodzi z głównej pętli obsługi zdarzeń, wywołując funkcję gtk_main_quit. Wywołanie tej funkcji jest konieczne do przerwania pracy aplikacji. Jeśli zamkniemy okno najwyższego poziomu, nie opuszczając pętli obsługi zdarzeń, wówczas program nadal będzie przetwarzał zdarzenia (dlatego w poprzednim przykładzie musieliśmy skorzystać ręcznie zatrzymać program). Funkcja zwrotna dla delete_event powinna zwracać wartość Boole'owską, która wskazuje, czy należy zapobiec zamknięciu okna. Zwrócenie wartości TRUE sprawi, że okno pozostanie otwarte, natomiast zwrócenie wartości FALSE zezwoli na zamknięcie okna. Jeśli wolno zamknąć okno, wówczas aplikacja otrzyma sygnał destroy, informujący ją o usuwaniu okna.

Drugim krokiem jest zarejestrowanie procedury obsługi zdarzenia w GTK+ przy pomocy funkcji gtk_signal_connect. Funkcja ta przyjmuje cztery parametry. Informują one GTK+ o tym, dla której kontrolki przeznaczona jest funkcja zwrotna, który sygnał jest obsługiwany przez funkcję, której funkcji należy użyć, kiedy zostanie wygenerowany sygnał, oraz

o dodatkowych parametrach, które należy przekazać do funkcji zwrotnej w momencie jej wywołania.

```
/* --- Wywołać funkcję usun_aplikacje, kiedy okno otrzyma sygnał
   --- "delete event" */
gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                    GTK_SIGNAL_FUNC (usun_aplikacje), NULL);
```

Zauważmy, że okno jest rzutowane przy pomocy makra GTK_OBJECT w wywołaniu funkcji gtk_signal_connect. Funkcja ta spodziewa się, że otrzyma parametr typu GtkObject a ponieważ GtkWindow wywodzi się w hierarchii obiektów od GtkObject, można rzutować go na GtkObject.

Powyższą linię kodu można zinterpretować następująco: jeśli w kontrolce okno wystąpi sygnał delete_event, wówczas należy wywołać funkcję usun_aplikacje z czwartym parametrem równym NULL.

Oto pełny program, z dodaną procedurą obsługi zdarzenia:

```
#include <gtk/gtk.h>

gint usun_aplikacje (GtkWidget *kontrolka, gpointer gdane)***
{

    g_print ("Kończę pracę...\n");
    gtk_main_quit ();

    /* --- Zezwalamy na zamknięcie okna, zwrócenie TRUE --- */
    /* --- zapobiegłoby jego zamknięciu */
    return (FALSE);
}

int main (int argc, char *argv[])
{
    GtkWidget *okno;

    /* --- inicjujemy GTK, przekazując argumenty wiersza polecenia */
    gtk_init (&argc, &argv);

    /* --- tworzymy okno; na razie NIE JEST ono widoczne --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    /* --- Kiedy okno otrzyma sygnał "delete_event", należy --- */
    /* --- wywołać funkcję usun_aplikacje --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
            GTK_SIGNAL_FUNC (usun_aplikacje), NULL);
```

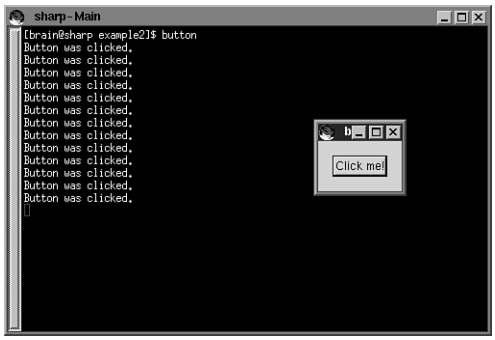
```
    /* --- teraz wyświetlamy okno --- */
    gtk_widget_show (okno);

    /* --- główna pętla zdarzeń w gtk. Funkcja nie powróci, dopóki */
    /* --- nie zostanie wywołana funkcja gtk_main_quit --- */
    gtk_main ();

    /* --- zwracamy kod wyjściowy --- */
    return 0;
}
```

Zauważmy, że podczas usuwania okna komunikat Kończę pracę... jest wyświetlany na konsoli, z której została uruchomiona aplikacja. Komunikat jest wyświetlany dzięki funkcji zwrotnej. Funkcja gtk_main pracuje w pętli, przetwarzając komunikaty aż do chwili, w której użytkownik zechce zamknąć okno. W takim przypadku wywoływana jest funkcja usun_aplikacje, aby powiadomić aplikację o wystąpieniu sygnału delete_ event, który został zarejestrowany przy pomocy funkcji gtk_signal_connect. Funkcja usun_aplikacje, wiedząc, że zamknięcie głównego okna powinno spowodować zakończenie aplikacji, wywołuje funkcję gtk_main_quit, która kończy pętlę gtk_main wywołaną przez aplikację. Program staje się dzięki niej prostą, ale poprawną aplikacją GTK+. We wcześniejszym przykładzie sygnał delete_event również był generowany, ale program nie posiadał procedury, która mogłaby go obsłużyć. W tym przypadku sygnał jest obsługiwany przez funkcję zwrotną, a aplikacja kończy działanie we właściwy sposób.

Dodawanie kontrolek

Proste okno, bez menu i kontrolek można nazwać aplikacją, choć nieszczególnie przydatną. Dodamy więc odrobinę funkcjonalności do pustego okna, które stworzyliśmy wcześniej; w tym przypadku będziemy dodawać do okna kontrolkę. Ponieważ utworzone okno najwyższego poziomu jest także pojemnikiem, można wewnątrz niego umieszczać kontrolki. Wykorzystamy funkcję gtk_button_new_with_label aby stworzyć przycisk, oraz gtk_container_add, aby umieścić przycisk wewnątrz okna. Zauważmy, że zarówno okno najwyższego poziomu jak i przycisk muszą zostać uwidocznione przy pomocy funkcji gtk_widget_show. Przycisk musi także w odrębnym kroku zostać dodany do pojemnika; przycisk nie pojawi się na ekranie, dopóki nie zostanie dodany do pojemnika, mimo stworzenia go i uwidocznienia. Kontrolki muszą być oknami najwyższego poziomu (bez rodzica), albo muszą znajdować się wewnątrz rodzica, aby można było je zobaczyć. Funkcja gtk_container_add ustawia relację

rodzic/potomek pomiędzy oknem najwyższego poziomu i kontrolką i umieszcza kontrolkę w oknie. Możemy także dodać do przycisku procedurę obsługi zdarzenia, obsługującą sygnał clicked. Po każdym kliknięciu przycisku myszą będzie wywoływana procedura kliknieto_przycisk, która wyświetli na konsoli odpowiedni komunikat. Rysunek 3.3 przedstawia przycisk, który wyświetla komunikat przy każdym kliknięciu.



Rysunek 3.3. Proste zdarzenia związane z przyciskiem GTK+

```
#include <gtk/gtk.h>

gint ZamknijOknoAplikacji (GtkWidget *kontrolka, gpointer gdane)
{
    g_print ("Kończę pracę...\n");
    gtk_main_quit ();

    /* --- Kontynuujemy zamykanie --- */
    return (FALSE);
}
/*
 * kliknieto_przycisk
 *
 * procedura obsługi zdarzenia, wywoływana po kliknięciu przycisku
 */

void kliknieto_przycisk (GtkWidget *kontrolka, gpointer gdane)
{
    g_print ("Kliknięto przycisk.\n";
}

int main (int argc, char *argv[])
```

```
{
    GtkWidget *okno;
    GtkWidget *przycisk;

    /* --- inicjujemy GTK, przekazując argumenty wiersza polecenia */
    gtk_init (&argc, &argv);

    /* --- tworzymy okno; na razie NIE JEST ono widoczne --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- funkcja wywoływana podczas zamykania aplikacji --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
            GTK_SIGNAL_FUNC (ZamknijOknoAplikacji), NULL);

    /* --- zostawiamy trochę miejsca dookoła obiektów w pojemniku  */
    gtk_container_border_width (GTK_CONTAINER (okno), 15);

    /* --- tworzymy przycisk --- */
    przycisk = gtk_button_new_with_label ("Kliknij mnie");

    /* --- przypisujemy do przycisku procedurę obsługi zdarzenia -- */
    /* --- przycisk będzie wywoływał funkcję kliknieto_przycisk --- */
    gtk_signal_connect (GTK_OBJECT (przycisk),
                                    "clicked",
                                    GTK_SIGNAL_FUNC (kliknieto_przycisk),
                                    NULL);

    /* --- główne okno zawiera przycisk --- */
    gtk_container_add (GTK_CONTAINER (okno), przycisk);

    /* --- uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* --- teraz uwidaczniamy okno --- */
    gtk_widget_show (okno);

    /* --- główna pętla zdarzeń w gtk. Funkcja nie powróci, dopóki */
    /* --- nie zostanie wywołana funkcja gtk_main_quit --- */
    gtk_main ();

    /* --- zwracamy kod wyjściowy --- */
    return 0;
}
```

Kontrolki dodaje się do okien w kilku krokach. Najpierw trzeba stworzyć kontrolkę i uzyskać wskaźnik GtkWidget.

Drugim krokiem jest ustawienie funkcji zwrotnych dla kontrolki. Niektóre kontrolki nie mogą mieć funkcji zwrotnych (na przykład etykieta), a niektóre nie muszą ich posiadać, ale większość kontrolek ma funkcje zwrotne, które umożliwiają interakcję z użytkownikiem.

Trzecim krokiem jest ustawienie właściwości kontrolki, np. wyrównanie kontrolki do lewej strony albo ustawienie etykiety na przycisku. Czasem właściwości mogą zostać ustawione w funkcji tworzącej kontrolkę; np. przycisk można utworzyć przy pomocy funkcji gtk_button_new, albo, wraz z etykietą, przy pomocy funkcji gtk_button_new_with_label. Ta druga funkcja jest wygodniejsza, niż oddzielne tworzenie przycisku i etykiety, a następnie dodawanie etykiety do przycisku.

Czwartym krokiem jest dodanie kontrolki do pojemnika. O ile kontrolka nie jest kontrolką najwyższego poziomu, to musi zostać dodana do widocznego pojemnika, zanim zostanie wyświetlona.

Ostatnim krokiem jest uwidocznienie kontrolki. Nie musi to być ostatni krok, ale zwykle tak się robi, ponieważ modyfikowanie właściwości widocznych kontrolek powoduje nieprzyjemne migotanie - zwłaszcza na wolniejszych komputerach, które nie potrafią szybko odświeżyć ekranu.

Wiele procedur obsługi zdarzeń

W wielu sytuacjach kontrolka potrzebuje kilku procedur obsługi, aby reagować na różne zdarzenia. Dodatkowy kod jest po prostu rozszerzeniem już istniejącego. W podrozdziale tym wzbogacimy poprzedni przykład o procedury obsługi zdarzeń, wywoływane w momencie przesunięcia wskaźnika myszy nad przycisk i odsunięcia go znad przycisku. Procedury te będą wyświetlać odpowiednie komunikaty na konsoli, w zależności od tego, czy wskaźnik myszy wkracza do przycisku, czy też go opuszcza.

```
/*
 * wskaznik_wchodzi
 *
 * Wywoływana w momencie, kiedy wskaźnik myszy przesuwa się nad
 * przycisk
 */
void wskaznik_wchodzi (GtkWidget *kontrolka, gpointer gdane);
{
    g_print ("Witamy w fantazyjnym przycisku\n");
```

```
}

/*
 * wskaznik_wychodzi
 *
 * Wywoływana w momencie, kiedy wskaźnik myszy odsuwa się znad
 * przycisku
 */
void wskaznik_wychodzi (GtkWidget *kontrolka, gpointer gdane);
{
    g_print ("Hej! Dokąd idziesz?\n");
}
```

Po napisaniu funkcji zwrotnych trzeba je zarejestrować w GTK+, aby zostały wywołane po wystąpieniu sygnału. Przypisujemy funkcje zwrotne do kontrolek przy pomocy funkcji gtk_signal_connect. Zdarzenie enter oznacza, że wskaźnik myszy przesunął się ponad przycisk, a zdarzenie leave oznacza, że wskaźnik myszy odsunął się znad przycisku. Przypisując te zdarzenia funkcjom możemy sprawić, że przycisk będzie reagował na trzy różne zdarzenia.

```
/* --- przypisujemy do przycisku procedurę obsługi zdarzenia -- */
/* --- przycisk będzie wywoływał funkcję wskaznik_wchodzi,  --- */
/* --- kiedy wskaźnik myszy przesunie się nad przycisk      --- */
gtk_signal_connect (GTK_OBJECT (przycisk),
                    "enter",
                    GTK_SIGNAL_FUNC (wskaznik_wchodzi),
                    NULL);

/* --- przypisujemy do przycisku procedurę obsługi zdarzenia -- */
/* --- przycisk będzie wywoływał funkcję wskaznik_wychodzi, --- */
/* --- kiedy wskaźnik myszy odsunie się znad przycisku      --- */
gtk_signal_connect (GTK_OBJECT (przycisk),
                    "leave",
                    GTK_SIGNAL_FUNC (wskaznik_wychodzi),
                    NULL);
```

Wskaźnik posiada obecnie trzy funkcje zwrotne, z których każda obsługuje inny sygnał i wyświetla odpowiedni komunikat. Możemy sprawdzić, jak generowane są sygnały i jak odpowiadają na nie funkcje zwrotne, przesuwając wskaźnik myszy nad przycisk i klikając go.

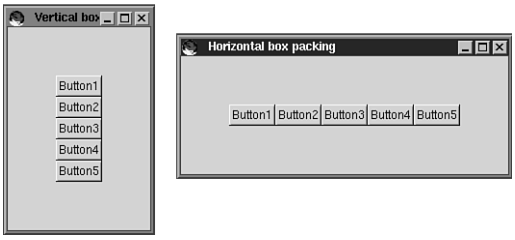
Pojemniki

Wiemy już, że okno najwyższego poziomu jest pojemnikiem, i że można umieszczać w nim kontrolki. Może wydać się to dziwne, ale pojemnikami jest bardzo wiele kontrolek. Ponieważ przycisk jest pojemnikiem, można w nim umieścić inną kontrolkę. W rzeczywistości, kiedy utworzymy przycisk z etykietą przy pomocy funkcji gtk_create_button_ with_label, wówczas kontrolka przycisku będzie zawierać kontrolkę etykiety.

Pojemniki w GTK+ mają jedno ograniczenie: mogą posiadać tylko jedną kontrolkę potomną. Dlatego, jeśli zmodyfikujemy poprzedni przykład tak, aby dodawał do okna macierzystego dwa przyciski, tylko pierwszy przycisk będzie widoczny. Aby obejść to ograniczenie, trzeba skorzystać z kontrolek pakujących.

Pola pakujące

Można dodać wiele kontrolek do pojemnika przy pomocy kontrolki pakującej, zwanej także polem pakującym. W przeciwieństwie do przycisków, kontrolki pakujące nie są wyświetlane na ekranie; są one niewidocznymi pojemnikami, które potrafią obsłużyć kilka kontrolek potomnych jednocześnie. Pionowe pole pakujące przechowuje kontrolki w układzie pionowym, a poziome - w układzie poziomym. Pionowe pola pakujące tworzymy przy pomocy funkcji gtk_vbox_new, a poziome przy pomocy gtk_hbox_new. Po stworzeniu pól można dodawać do nich kontrolki, używając funkcji gtk_box_pack_start i gtk_box_pack_end. Funkcja gtk_box_pack_start umieszcza kontrolki w polu pakującym na górze (dla pola pionowego) albo po lewej stronie (dla pola poziomego). Funkcja gtk_box_pack_end umieszcza kontrolki w polu pakującym na dole (dla pola pionowego) albo po prawej stronie (dla pola poziomego). Aby kontrolki zostały wyświetlone, muszą zostać dodane do pola pakującego, a samo pole musi zostać dodane do pojemnika; dzięki temu pojemnik będzie mógł przechowywać wiele kontrolek. Pojemnik i pole pakujące muszą zostać uwidocznione, chociaż pole pakujące nie zawiera żadnych widzialnych składników. Widoczność ma wpływ na kontrolki, dodane do pojemnika. Rysunek 3.4 przedstawia kontrolki umieszczone w pionowym i poziomym polu pakującym.



Rysunek 3.4. Działanie pionowych i poziomych pól pakujących.

Funkcje tworzące pola pakujące przyjmują dwa parametry, które określają, jak pole będzie wyglądało na ekranie.

```
/* --- Tworzymy poziome pole pakujące --- */
xpole = gtk_hbox_new (bRownomierne, nOdstep);

/* --- Tworzymy pionowe pole pakujące --- */
ypole = gtk_vbox_new (bRownomierne, nOdstep);
```

Pierwszy parametr, rownomierne, określa, czy wszystkie dodawane kontrolki będą zajmować taki sam obszar na ekranie. Jeśli na przykład dodalibyśmy do poziomego pola pakującego pięć przycisków, o etykietach Jacek, Jan, Ania, Asia i Aleksander Wielki, wówczas nierównomierne pole pakujące stworzyłoby przyciski o takiej wielkości, jaka byłaby potrzebna do wyświetlenia tekstu na przycisku. Natomiast w metodzie równomiernej zostałby obliczony rozmiar najszerszego przycisku (łatwo domyślić się, o który chodzi), i każdy z przycisków zajmowałby właśnie taki obszar, niezależnie od tego, czy tekst wypełniałby go całkowicie. Drugi parametr, odstep, określa, ile miejsca należy pozostawić pomiędzy kontrolkami umieszczanymi w polu pakującym. Ustawienie go na zero powoduje, że pomiędzy kontrolkami nie będzie żadnych odstępów.

Funkcja gtk_box_pack_start przyjmuje trzy parametry, które określają, jak należy umieścić pojedynczą kontrolkę wewnątrz pola pakującego.

```
gtk_box_pack_start (pole, kontrolka, rozszerzanie, wypelnienie, odstep);
```

Znacznik rozszerzanie określa, czy pole wokół kontrolek może zostać rozszerzone tak, aby wypełnić przestrzeń pozostałą po dodaniu wszystkich kontrolek do pola pakującego. Znacznik rozszerzanie jest ignorowany, jeśli pole pakujące zostało utworzone z ustawionym znacznikiem rownomierne, ponieważ znacznik ten wskazuje, że kontrolki powinny zajmować taki sam obszar wewnątrz pola pakującego.

Znacznik wypelnienie określa, czy dodatkowa przestrzeń wokół danej kontrolki powinna zostać przez nią zajęta. Ustawienie go na TRUE po-

zwala kontrolce na zwiększenie swoich rozmiarów tak, aby zajęła cały obszar przydzielony jej przez pole pakujące. Ustawienie go na FALSE zmusza kontrolkę do wykorzystania tylko takiego obszaru, jakiego potrzebuje, a dodatkowa przestrzeń będzie otaczać kontrolkę.

Znacznik odstep określa, ile pikseli wypełnienia powinno otaczać kontrolkę. Zazwyczaj ustawiany jest na zero.

Poniższy przykład przedstawia okno z pionowym polem pakującym, wypełnionym przez pięć przycisków. Kod służący do tworzenia i dodawania przycisków został umieszczony w funkcji o nazwie UpakujNowyPrzycisk, w celu uproszczenia programu.

```
/* polepak.c
 *
 * Program ilustrujący różnicę pomiędzy poziomymi i pionowymi polami
 * pakującymi. Wyświetla dwa okna z odmiennie upakowanymi przyciskami
 */

#include <gtk/gtk.h>

/*
 * UpakujNowyPrzycisk
 *
 * Tworzy przycisk, pakuje go do określonego pola i uwidacznia go
 *
 * pole - pole pakujące, do którego należy dodać przycisk
 * szEtykieta - etykieta na przycisku
 * zwraca utworzoną kontrolkę przycisku
 */

GtkWidget *UpakujNowyPrzycisk (GtkWidget *pole, char *szEtykieta)
{
    GtkWidget *przycisk;

    /* --- Tworzymy nowy przycisk --- */
    przycisk = gtk_button_new_with_label (szEtykieta);

    /* --- Upakowujemy przycisk w polu --- */
    gtk_box_pack_start (GTK_BOX (pole), przycisk, FALSE, FALSE, 0);

    /* --- Uwidaczniamy przycisk --- */
    gtk_widget_show(przycisk);

    return (przycisk);
```

```
}

gint Usun (GtkWidget *kontrolka, gpointer *dane)
{
    gtk_main_quit ();

    /* --- Kontynuujemy zamykanie --- */
    return (FALSE);
}

int main (int argc, char *argv[])
{
    GtkWidget *okno;
    GtkWidget *przycisk;
    GtkWidget *pole;

    /* --- Inicjacja GTK --- */
    gtk_init (&argc, &argv);

    /* ------------------------------------------------ */
    /* --- Tworzymy pionowe pole pakujące --- */
    /* ------------------------------------------------ */

    /* --- tworzymy okno najwyższego poziomu --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- nadajemy oknu tytuł --- */
    gtk_window_set_title (GTK_WINDOW (okno), "Pionowe pole pakujące");

    /* --- chcemy wiedzieć, kiedy okno będzie zamykane --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                        GTK_SIGNAL_FUNC (Usun), NULL);

    /* --- tworzymy obramowanie okna --- */
    gtk_container_border_width (GTK_CONTAINER (okno), 50);

    /* --- Tworzymy pionowe pole pakujące --- */
    pole = gtk_vbox_new (FALSE, 0);

    /* --- Tworzymy przyciski --- */
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk1");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk2");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk3");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk4");
```

```
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk5");

    /* --- uwidaczniamy główne okno --- */
    gtk_container_add (GTK_CONTAINER (okno), pole);
    gtk_widget_show (pole);
    gtk_widget_show (okno);

    /* ------------------------------------------------ */
    /* --- Tworzymy poziome pole pakujące --- */
    /* ------------------------------------------------ */

    /* --- tworzymy okno najwyższego poziomu --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- nadajemy oknu tytuł --- */
    gtk_window_set_title (GTK_WINDOW (okno), "Poziome pole pakujące");

    /* --- tworzymy obramowanie okna --- */
    gtk_container_border_width (GTK_CONTAINER (okno), 50);

    /* --- Tworzymy poziome pole pakujące --- */
    pole = gtk_hbox_new (FALSE, 0);

    /* --- Tworzymy przyciski --- */
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk1");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk2");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk3");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk4");
    przycisk = UpakujNowyPrzycisk (pole, "Przycisk5");

    /* --- uwidaczniamy główne okno --- */
    gtk_container_add (GTK_CONTAINER (okno), pole);
    gtk_widget_show (pole);
    gtk_widget_show (okno);

    /* --- Uruchamiamy pętlę obsługi zdarzeń --- */
    gtk_main ();

    exit (0);
}
```

Pola pakujące można umieszczać także wewnątrz innych pól pakujących, tworząc ekran wypełniony kontrolkami w układzie innym niż tylko poziomy lub pionowy. Zazwyczaj umieszcza się rząd kontrolek w pozio-

mym polu pakującym, a następnie umieszcza pole poziome (wraz z innymi polami poziomymi) w polu pionowym. Metoda ta pozwala na elastyczne zorganizowanie kontrolek na ekranie.

Tabele pakujące

Tabele pakujące przypominają pola pakujące, ponieważ również pozwalają ma umieszczenie wielu kontrolek w oknie. Pola pakujące umożliwiają jednakże tylko jednowymiarowe rozmieszczanie kontrolek, natomiast tabele pakujące przypominają tabele HTML: przy ich pomocy można rozmieszczać kontrolki w rzędach i kolumnach, przy czym obiekty mogą zajmować kilka rzędów lub kolumn. Funkcja tworząca tabele pakujące wygląda następująco:

```
tabela = gtk_table_new (rzedy, kolumny, rownomierne);
```

Pierwsze dwa argumenty określają liczbę rzędów i kolumn w tabeli, które muszą być znane w momencie tworzenia tabeli. Trzeci argument, rownomierne, daje takie same efekty, jak w przypadku pól pakujących: jeśli jest ustawiony na TRUE podczas tworzenia tabeli, wówczas komórki tabeli mają rozmiar największej przechowywanej w niej kontrolki. Jeśli znacznik rownomierne jest ustawiony na FALSE, wówczas szerokość każdej kolumny będzie równa szerokości największej kontrolki w tej kolumnie, a wysokość każdego rzędu będzie równa wysokości największej kontrolki w tym rzędzie.

Kolumny są ponumerowane od 0 do liczba_kolumn - 1, natomiast rzędy od 0 do liczba_rzędów - 1.

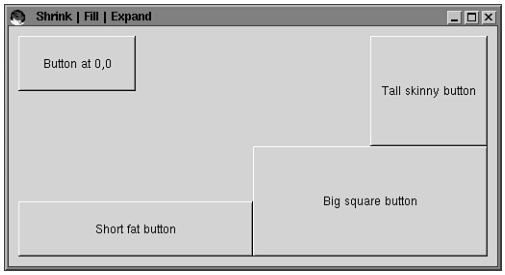
Podczas dodawania kontrolek należy określić początkową kolumnę, końcową kolumnę, początkowy rząd i końcowy rząd. Jeśli kontrolka ma być umieszczona w lewym górnym rogu tabeli, wówczas zakres kolumn będzie wynosił od 0 do 1, a zakres rzędów również od 0 do 1. Parametry xodstep i yodstep określają, ile pikseli wypełnienia pozostawić dookoła kontrolki.

Dwie funkcje, służące do dodawania kontrolek do tabeli, to:

```
gtk_table_attach (tabela,
                  potomek,
                  lewo,
                  prawo,
                  gora,
                  dol,
                  xopcje,
                  yopcje,
```

```
                    xodstep,
                    yodstep);

    gtk_table_attach_defaults (tabela,
                               kontrolka,
                               lewo,
                               prawo,
                               gora,
                               dol);
```

Obie funkcje dodają kontrolki do tabeli. Funkcja gtk_table_attach_defaults przyjmuje mniej parametrów, korzystając z domyślnych ustawień dla opcji i odstępu, używanych w funkcji gtk_table_attach. Ustawienia domyślne dla parametrów xodstep i yodstep mają wartość zero, a dla xopcje i yopcje wartość (GTK_FILL | GTK_EXPAND). Kolejny przykład pokazuje różnice pomiędzy opcjami na przykładzie tabeli 4 x 4 (patrz rysunek 3.5). Każde z okien zawiera cztery przyciski, dodane przy różnych ustawieniach znaczników. Kiedy rozmiar okna ulegnie zmianie, kontrolki zostaną odpowiednio przerysowane, na podstawie wartości znaczników użytych podczas dodawania kontrolek do okna.



Rysunek 3.5. Opcje tabeli pakującej.

Opcje xopcje i yopcje mogą przyjmować wartości GTK_FILL, GTK_SHRINK, GTK_EXPAND albo ich dowolną kombinację. GTK_FILL określa, że kontrolka powinna się rozszerzać, aby zająć cały wyznaczony jej obszar. GTK_SHRINK pozwala na skurczenie się kontrolki do mniejszego rozmiaru, niż został jej początkowo przydzielony. GTK_EXPAND powoduje rozszerzenie się tabeli w taki sposób, aby zajmowała całe miejsce w oknie, do którego została wstawiona.

W przykładowym programie przycisk w lewym górnym rogu zajmuje jedną komórkę tabeli (0-1, 0-1), przyciski w prawym górnym i lewym dolnym rogu zajmują dwie komórki, a przycisk w prawym dolnym rogu

zajmuje cztery komórki. Można zaobserwować działanie różnych atrybutów, zmieniając rozmiary okien. Cały kod służący do tworzenia okien został umieszczony w funkcji UtworzTabele.

```
/* tabela
 *
 * Ilustruje sposób użycia tabel pakujących
 *
 */

#include <gtk/gtk.h>

/*
 * Usun
 *
 * Wywoływana w momencie zamykania głównego okna. Umożliwia powrót z
 * wywołania funkcji gtk_main()
 */

gint Usun (GtkWidget *kontrolka, gpointer *dane)
{
    gtk_main_quit();

    /* --- Kontynuujemy zamykanie --- */
    return (FALSE);
}

/*
 * UtworzTabele
 *
 * Tworzy okno najwyższego poziomu z kilkoma opcjami. Ponieważ trzeba
 * będzie to zrobić kilka razy, umieszczamy właściwy kod w funkcji
 */
UtworzTabele (char *szTytul, gint xopcje, gint yopcje)
{
    GtkWidget *okno;
    GtkWidget *przycisk;
    GtkWidget *tabela;
    /* --- tworzymy okno najwyższego poziomu --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    /* --- nadajemy oknu tytuł --- */
    gtk_window_set_title (GTK_WINDOW (okno), szTytul);
    /* --- tworzymy obramowanie okna --- */
```

```
    gtk_container_border_width (GTK_CONTAINER (okno), 10);
    /* --- tworzymy tabelę 4x4 --- */
    tabela = gtk_table_new (4, 4, TRUE);
    /* --- dodajemy przycisk 1x1 w pozycji 0, 0 --- */
    przycisk = gtk_button_new_with_label ("Przycisk 0,0");
    gtk_table_attach (GTK_TABLE (tabela), przycisk, 0, 1, 0, 1,
                xopcje, yopcje, 0, 0);
    gtk_widget_show (przycisk);
    /* --- dodajemy przycisk 2x2 w pozycji 2-4, 2-4 --- */
    przycisk = gtk_button_new_with_label("Duży, kwadratowy przycisk");
    gtk_table_attach (GTK_TABLE (tabela), przycisk, 2, 4, 2, 4,
                        xopcje, yopcje, 0, 0);
    gtk_widget_show (przycisk);

    /* --- dodajemy niski, szeroki przycisk 2x1 --- */
    przycisk = gtk_button_new_with_label ("Niski, szeroki przycisk");
    gtk_table_attach (GTK_TABLE (tabela), przycisk, 0, 2, 3, 4,
                        xopcje, yopcje, 0, 0);
    gtk_widget_show (przycisk);

    /* --- dodajemy wysoki, wąski przycisk 1x2 --- */
    przycisk = gtk_button_new_with_label ("Wysoki, wąski\nprzycisk");
    gtk_table_attach (GTK_TABLE (tabela), przycisk, 3, 4, 0, 2,
                        xopcje, yopcje, 0, 0);

    /* --- Uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* --- Dodajemy tabelę do okna --- */
    gtk_container_add (GTK_CONTAINER (okno), tabela);

    /* --- Uwidaczniamy tabelę --- */
    gtk_widget_show (tabela);

    /* --- Uwidaczniamy okno --- */
    gtk_widget_show (okno);
}

/*
 * --- Main
 *
 * Tutaj zaczyna się program
 *
```

```
 * Zamiast powielać ten sam kod, wywołujemy funkcję UtworzTabele.
 * Wykona ona za nas całą pracę - musimy tylko przekazać do niej
 * parametry, określające wygląd okna
 */
int main (int argc, char *argv[])
{
    /* --- Inicjacja GTK --- */
    gtk_init (&argc, &argv);

    /* --- Bez ustawionych znaczników --- */
    UtworzTabele ("Bez znaczników", 0, 0);

    /* --- Zaczynamy tworzyć okna ze znacznikami --- */
    UtworzTabele ("FILL", GTK_FILL, GTK_FILL);

    UtworzTabele ("SHRINK", GTK_SHRINK, GTK_SHRINK);

    UtworzTabele ("EXPAND", GTK_EXPAND, GTK_EXPAND);

    UtworzTabele ("EXPAND | SHRINK",
                  GTK_EXPAND | GTK_SHRINK,
                  GTK_EXPAND | GTK_SHRINK);

    UtworzTabele ("FILL | EXPAND",
                  GTK_FILL | GTK_EXPAND,
                  GTK_FILL | GTK_EXPAND);

    UtworzTabele ("SHRINK | FILL",
                  GTK_SHRINK | GTK_FILL,
                  GTK_SHRINK | GTK_FILL);

    UtworzTabele ("SHRINK | FILL | EXPAND",
                  GTK_SHRINK | GTK_FILL | GTK_EXPAND,
                  GTK_SHRINK | GTK_FILL | GTK_EXPAND);

    /* --- Uruchamiamy pętlę gtk --- */
    gtk_main ();

    return (0);
}
```

Podsumowanie

Rozdział ten omówił podstawy programowania GTK+, przedstawiając prostą aplikację, która zilustrowała strukturę programu GTK+. Opisał także podstawowe zagadnienia, związane ze zdarzeniami i kontrolkami GTK+, oraz niektóre typy danych GTK+. Informacje te stanowią fundament dla następnego rozdziału - „Podstawowe kontrolki".