

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTEL尼亚

FRAGMENTY KSIĄŻEK ONLINE

MySQL. Budowanie interfejsów użytkownika. Vademecum profesjonalisty

Autor: Matthew Stucky

Tłumaczenie: Tomasz Miszkiel

ISBN: 83-7197-885-5

Tytuł oryginału: [MySQL Building User Interfaces](#)

Format: B5, stron: 582

[Przykłady na ftp: 1098 kB](#)



MySQL jest szybkim, przenośnym i – co najważniejsze – darmowym systemem bazodanowym, który zdobył ogromną popularność w zastosowaniach internetowych. Książka „MySQL. Budowanie interfejsów użytkownika.” pokaże Ci jak używać tej bazy także poza Internetem. Nauczysz się z niej budowania atrakcyjnych interfejsów użytkownika współpracujących z tą bazą. W tym celu wykorzystana zostanie biblioteka GTK+.

Książka jest przeznaczona głównie dla programistów, którzy rozważają wykorzystanie Linuksa w tworzonych przez siebie rozwiązaniach, dla osób posiadających pewne doświadczenie w tworzeniu systemów opartych na architekturze klient-serwer (np. za pomocą Visual Basic a i SQL Server). Aby w pełni skorzystać z informacji zawartych w książce, wystarczy znać język C i podstawy programowania baz danych.

- Poznasz mocne strony i zalety bazy MySQL
- Nauczysz się tworzyć programy w systemie Linuks, a następnie, korzystając z ich kodu źródłowego, kompilować je w systemie Windows.
- Dowiesz się, w jaki sposób można instalować programy korzystając ze skryptów lub pakietów RPM (RedHat Package Manager).
- Nauczysz się korzystać z interfejsu języka C dla MySQL w aplikacjach, w których użyto bibliotekę GTK+.
- Poznasz sposoby wyświetlania danych pochodzących z MySQL w obiektach GTK+.
- Będziesz w stanie utworzyć (w języku C) aplikacje, które wyglądem i zachowaniem przypominają programy Windowsowe
- Utworzysz aplikacje, które będą dynamicznie tworzyć graficzny interfejs użytkownika, dzięki czemu będziesz mógł zmienić wygląd interfejsu bez konieczności rekompilacji kodu źródłowego
- Prześledzisz krok po kroku proces powstawania kompletnej aplikacji korzystającej z MySQL.



Spis treści

O Autorze	9
O Recenzentach technicznych	10
Wprowadzenie	11
Dla kogo przeznaczona jest ta książka?	12
Układ książki	12
Konwencje przyjęte w książce	13
Część I Szybkie wprowadzenie	15
Rozdział 1. MySQL dla administratorów i użytkowników Access i SQL Server	17
Dlaczego właśnie MySQL?	17
Licencje	18
Porównanie typów danych MySQL	
z typami dostępnymi w Access 2000 i SQL Server 7	19
Czego brakuje w MySQL?	21
MySQL, MYSQL i mysql — trzy podobne słowa, trzy różne znaczenia	27
Tylko dla użytkowników programu Access: demony i usługi	29
Skąd pobrać pakiet instalacyjny MySQL?	30
Decyzja dotycząca metody instalacji — wszystkie za i przeciw	31
Pliki potrzebne do instalacji RPM	33
Instalacja	34
Pierwsze spotkanie z MySQL: tworzenie, wykorzystywanie i usuwanie bazy danych	38
Niestandardowe, unikatowe konstrukcje SQL wykorzystywane w MySQL	
(rozszerzenie standardu ANSI SQL92)	40
Programy użytkowe w MySQL	45
Interfejs API języka C dla MySQL	53
Rozdział 2. GTK+ dla programistów korzystających z Visual Basic	59
Dlaczego powinniśmy korzystać z GTK+?	60
Gdzie zdobyć GTK+ i jak dokonać jego instalacji?	61
Licencje	61
Ogólny opis biblioteki GTK+	63
Pojemniki GTK+ jako odpowiedniki kontrolki zmieniających rozmiary	71
Podstawowe widgety w programie	73
Rozdział 3. Więcej widgetów GTK+	95
Widget GtkCombo	95
GtkProgressBar i GtkStatusBar	99
GtkFrame i GtkAspectFrame	106
Okna dialogowe i informacyjne, czyli obiekty GtkDialog, GtkFileSelection,	
GtkColorSelectionDialog i GtkFontSelectionDialog	110
Systemy menu: obiekty menu, „fabryka elementów” i elementy menu kontekstowego	117

Rozdział 4.	Zaawansowane obiekty GTK+	127
	GtkTable	127
	GtkTree i GtkCTree	132
	GtkFixed	137
	GtkLayout	140
	GtkScrolledWindow	142
	GtkNotebook	144
	GtkPaned	147
Rozdział 5.	Glade dla programistów korzystających z Visual Basic	151
	Środowisko Glade	151
	Pierwsze spotkanie z Glade: aplikacja WitajSwiecie	153
	Szczegółowa analiza wybranych plików projektu WitajSwiecie	162
	Glade i widgety	170
	Tworzenie projektu: trzeba poznać hierarchię obiektów	173
	Komunikacja pomiędzy oknami w projekcie utworzonym w Glade	174
	Zmienne o zasięgu globalnym a poprawne pisanie programów	180
Część II	Prawdziwe aplikacje	183
Rozdział 6.	Projektowanie i zasada działania aplikacji przyjęcia zamówienia	185
	Projekt aplikacji	186
	Interfejs użytkownika	189
	Tworzenie bazy danych	193
	Wdrożenie aplikacji	195
	Modernizacja aplikacji	199
Rozdział 7.	Utworzenie aplikacji do obsługi zamówień dla SESI	201
	Tworzenie interfejsu użytkownika za pomocą Glade	202
	Funkcje aplikacji	214
	Połączenie interfejsu i funkcji wykonujących poszczególne zadania programu	248
	Kompilacja programu	259
	Uwagi dotyczące utworzonego programu	259
Rozdział 8.	Aplikacja Wyznaczanie prowizji — projekt i analiza	263
	Przedstawienie problemu	264
	Interfejs użytkownika	267
	Baza danych, przygotowanie autoryzacji	271
	Tworzenie bazy danych	273
Rozdział 9.	Konstrukcja aplikacji Prowizje	281
	Budowanie interfejsu użytkownika za pomocą Glade	281
	Kontrola wyświetlanych danych	286
	Funkcje tworzonej aplikacji	289
	Powiązanie funkcji „narzędziowych” z obiektami interfejsu użytkownika — funkcje zwrotne w pliku callbacks.c	318
	Uwagi dotyczące utworzonego programu	325
Rozdział 10.	Instalacja aplikacji Wyznaczanie prowizji	327
	Kompilacja z wiersza poleceń	327
	Korzyści płynące z zastosowania Make	328
	Instalowanie aplikacji	330
	Kompilacja programu dla Win32 za pomocą MS Visual C++	341

Rozdział 11. Projekt aplikacji Raporty	351
Definicja problemu.....	351
Interfejs użytkownika	352
Baza danych, uprawnienia użytkowników	356
Tworzenie bazy danych.....	357
Rozdział 12. Konstrukcja aplikacji Raporty	361
Opis problemu.....	361
Aplikacja sterująca: Raporty.....	361
Raport 1. Tabela.....	371
Raport 2. Wykres kołowy	377
Raport 3. Wykres liniowo-kolumnowy	387
Raport 4. Wykres punktowy.....	393
Rozdział 13. Kompilacja i instalacja aplikacji Raporty	403
Kompilacja i dystrybucja w formie pliku RPM	403
Kompilacja programów za pomocą MinGW dla Win32	408
Część III Przykład	415
Rozdział 14. Dynamiczne tworzenie interfejsu użytkownika za pomocą XML....	417
Jeszcze raz program „Witaj Świecie”	417
Komunikacja pomiędzy oknami.....	422
Dokonywanie zmian w aplikacji bez konieczności ponownej kompilacji	430
Dodatki	433
Dodatek A Pliki wygenerowane przez Glade — aplikacja dla SESI.....	435
interface.c.....	435
sesi.glade.....	460
Dodatek B Pliki wygenerowane przez Glade — aplikacja Prowizje.....	499
interface.c.....	499
prowizje.glade.....	516
Dodatek C Pliki wygenerowane przez Glade — aplikacja Raporty	547
Pliki dla programu głównego.....	547
Pliki dla programu tworzącego raport w formie tabeli.....	551
Pliki dla programu tworzącego wykres kołowy	556
Pliki dla programu tworzącego raport w postaci wykresu liniowo-kolumnowego	559
Pliki dla programu generującego raport w postaci wykresu punktowego.....	562
Skorowidz	571

Rozdział 4.

Zaawansowane obiekty GTK+

W tym rozdziale opisano zaawansowane obiekty GTK+, służące do rozmieszczania i prezentacji innych elementów. Szczegółowo opisano tu następujące widgety:

- `GtkTable`,
- `GtkTree` i `GtkCTree`,
- `GtkFixed`,
- `GtkLayout`,
- `GtkScrolledWindow`,
- `GtkNotebook`,
- `GtkPaned`.

GtkTable

Obiekt `GtkTable` działa na zasadzie siatki, w której można umieszczać inne obiekty. Można powiedzieć, że `GtkTable` jest połączeniem dwóch prostokątnych pojemników (poziomego i pionowego). Na rysunku 4.1 przedstawiono obiekt `GtkTable`, który utworzymy w przykładowym programie (listing 4.1). W `GtkTable` można umieścić obiekt w kilku komórkach (poziomo lub pionowo).

Przykładowy program przedstawia działanie obiektu `GtkTable`. Pokazano tu również nowy trend tworzenia „programów w akcji”, gdzie w odpowiedzi na wystąpienie zdarzeń tworzy się (lub niszczy) określone widgety. Należy pamiętać, że jeśli będziemy korzystać z tej metody w tworzonych aplikacjach, to jednocześnie będziemy musieli znaleźć sposób na przechowanie danych, które mają być przekazane tworzonemu na nowo obiektom.

Rysunek 4.1 przedstawia okno utworzonej aplikacji, której kod źródłowy znajduje się w listingu 4.1.

Rysunek 4.1.
*Uruchomiony
 przykładowy program*



Listing 4.1. Prezentacja obiektu GtkTable

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

GtkWidget *frm_table;
GtkWidget *tbl;

/* Poniższe przyciski zostaną umieszczone w GtkTable
 * w 1. kolumnie. Pierwszy z nich będzie znajdował się w 1. rzędzie
 * 1. kolumny, drugi w 2. rzędzie 1. kolumny i tak dalej.
 */

GtkWidget *cmd_1_1;
GtkWidget *cmd_2_1;
GtkWidget *cmd_4_1;
GtkWidget *cmd_1_2;
GtkWidget *cmd_1_3;
GtkWidget *cmd_3_2;
GtkWidget *cmd_2_3;
GtkWidget *cmd_3_3;
GtkWidget *chk_1;

void destroy_main();
void make_table_buttons();
void attach_widgets();
void cmd_1_1_clicked();
void cmd_2_1_clicked();
void cmd_4_1_clicked();
void cmd_1_2_clicked();
void cmd_1_3_clicked();
void cmd_3_2_clicked();
void cmd_2_3_clicked();
void cmd_3_3_clicked();

gint main(gint argc, gchar *argv[])
{
    gtk_init(&argc, &argv);

    frm_table = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```

gtk_window_set_title(GTK_WINDOW(frm_table), "Prezentacja tablicy obiektow");

gtk_signal_connect(GTK_OBJECT(frm_table),
    "destroy",
    GTK_SIGNAL_FUNC(destroy_main),
    NULL);

make_table_buttons();

/* Inicjalizacja przyciskow z uzyciem wartosci domyslnych.
 *
 * Skorzystamy tu z gtk_table_attach_defaults() zamiast
 * z funkcji attach_widgets(), w celu demonstracji dzialania
 * funkcji gtk_table_attach_defaults().
 */

gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_1_1, 0, 1, 0, 1);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_2_1, 0, 1, 1, 3);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_4_1, 0, 4, 3, 4);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_1_2, 1, 2, 0, 2);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_1_3, 2, 3, 0, 1);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_3_2, 1, 2, 2, 3);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_3_3, 2, 3, 2, 3);
gtk_table_attach_defaults(GTK_TABLE(tbl), cmd_2_3, 2, 4, 1, 2);

chk_1 = gtk_check_button_new_with_label("Przycisk wyboru");

gtk_table_attach_defaults(GTK_TABLE(tbl), chk_1, 3, 4, 0, 1);

gtk_container_add(GTK_CONTAINER(frm_table), tbl);

gtk_widget_show_all (frm_table);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

void attach_widgets(gint _OPTIONS, gint _PACKING)
{
    gtk_table_attach(GTK_TABLE(tbl), cmd_1_1, 0, 1, 0, 1, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_2_1, 0, 1, 1, 3, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_4_1, 0, 4, 3, 4, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_1_2, 1, 2, 0, 2, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_1_3, 2, 3, 0, 1, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_3_2, 1, 2, 2, 3, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), cmd_3_3, 2, 3, 2, 3, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
}

```

```

    gtk_table_attach(GTK_TABLE(tbl), cmd_2_3, 2, 4, 1, 2, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
    gtk_table_attach(GTK_TABLE(tbl), chk_1, 3, 4, 0, 1, _OPTIONS, _OPTIONS, _PACKING,
        _PACKING);
}

void make_table_buttons()
{
    tbl = gtk_table_new(4, 4, TRUE);

    /* Parametry powyższej funkcji to: rzedy, kolumny, oznaczenie jednorodności.
    */

    cmd_1_1 = gtk_button_new_with_label("Wypełnienie 9 obiektami");
    gtk_signal_connect(GTK_OBJECT(cmd_1_1),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_1_1_clicked),
        NULL);
    cmd_2_1 = gtk_button_new_with_label("Ustaw\nopcje\nFILL");
    gtk_signal_connect(GTK_OBJECT(cmd_2_1),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_2_1_clicked),
        NULL);
    cmd_4_1 = gtk_button_new_with_label("Wybierz opcje SHRINK");
    gtk_signal_connect(GTK_OBJECT(cmd_4_1),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_4_1_clicked),
        NULL);
    cmd_1_2 = gtk_button_new_with_label("Wybierz\nOpcje\nEXPAND");
    gtk_signal_connect(GTK_OBJECT(cmd_1_2),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_1_2_clicked),
        NULL);
    cmd_1_3 = gtk_button_new_with_label("Wybierz\nOpcje\nFILL\ni\nSHRINK");
    gtk_signal_connect(GTK_OBJECT(cmd_1_3),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_1_3_clicked),
        NULL);
    cmd_3_2 = gtk_button_new_with_label("Wybierz\nOpcje\nSHRINK\ni\nEXPAND");
    gtk_signal_connect(GTK_OBJECT(cmd_3_2),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_3_2_clicked),
        NULL);
    cmd_3_3 = gtk_button_new_with_label("Wybierz\nopcje\nFILL\ni\nEXPAND");
    gtk_signal_connect(GTK_OBJECT(cmd_3_3),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_3_3_clicked),
        NULL);
    cmd_2_3 = gtk_button_new_with_label("Wybierz opcje\nFILL, EXPAND i SHRINK");
    gtk_signal_connect(GTK_OBJECT(cmd_2_3),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_2_3_clicked),
        NULL);
    chk_1 = gtk_check_button_new_with_label("Przycisk wyboru");
}

```

```
void cmd_1_1_clicked()
{
    /* Tu wykorzystamy po raz pierwszy funkcje gtk_widget_destroy.
     * Po naciśnięciu przycisku przez użytkownika będziemy
     * niszczyć i tworzyć od nowa całą tablicę z obiektami.
     * Musimy tak postąpić, bo obiekt GtkTable nie posiada funkcji
     * *_set_*.
     *
     * Nawet gdyby taka instrukcja istniała, można by i tak wykorzystać
     * omawiany sposób. Wpisywanie podobnych funkcji dla każdego przycisku
     * jest procesem znużającym, ale jeśli samodzielnie wykonamy zmiany
     * rozmieszczenia obiektów w interfejsie, to nauczymy się zarządzać
     * powierzchnią tablicy i będziemy mieć nad nią pełną kontrolę.
     */

    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(0, 9);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

void cmd_2_1_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_FILL, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

void cmd_4_1_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_SHRINK, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

void cmd_1_2_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_EXPAND, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

void cmd_1_3_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_FILL | GTK_SHRINK, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
}
```

```

    gtk_widget_show_all (frm_table);
}

void cmd_3_2_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_SHRINK | GTK_EXPAND, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

void cmd_3_3_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_FILL | GTK_EXPAND, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

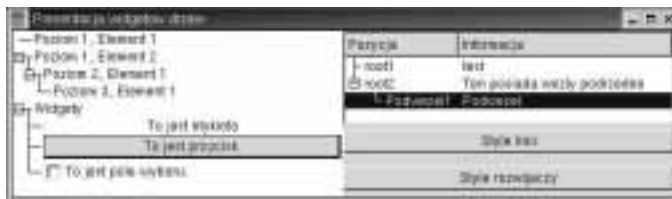
void cmd_2_3_clicked()
{
    gtk_widget_destroy(tbl);
    make_table_buttons();
    attach_widgets(GTK_SHRINK | GTK_FILL | GTK_EXPAND, 0);
    gtk_container_add(GTK_CONTAINER(frm_table), tbl);
    gtk_widget_show_all (frm_table);
}

```

GtkTree i GtkCTree

Obiekty `GtkTree` i `GtkCTree` służą do prezentacji drzewa elementów. Litera „C” w nazwie `GtkCTree` oznacza „kolumny” (C pochodzi od angielskiego słowa *Columns*); można powiedzieć, że `GtkCTree` jest skrzyżowaniem dwóch, znanych z VB kontroltek: kontrolki prezentującej dane i kontrolki drzewa (rysunek 4.2, listing 4.2). W GTK+ obiekt `GtkContainer` jest obiektem nadrzędnym dla `GtkList`, `GtkCList` i `GtkTree`. `GtkCTree` pochodzi od `GtkCList`, ale podobny związek nie istnieje pomiędzy `GtkTree` i `GtkList`.

Rysunek 4.2.
*Drzewa elementów,
czyli GtkTree
i GtkCTree*



W tym programie porównano cechy i wygląd widgetów `GtkTree` (po lewej stronie okna) i `GtkCTree` (po prawej stronie okna). Główną różnicą pomiędzy tymi obiektami jest fakt, że `GtkCTree` akceptuje tylko mapy pikselowe i tekst, a w `GtkTree` można umieścić dowolne widgety potomne.

Listing 4.2. *GtkTree i GtkCTree*

```

#include <gtk/gtk.h>

GtkWidget *frm_trees;
GtkWidget *tree_left, *ctree_right;
GtkWidget *subtree_item2;
GtkWidget *subtree_item3;
GtkWidget *subtree_widgets;

void destroy_main();
void cmd_cycle_line_style_clicked();
void cmd_cycle_expander_clicked();

gint main(gint argc, gchar *argv[])
{
    GtkWidget *hbox_main, *vbox_left, *vbox_right;
    GtkWidget *lvl1_item1, *lvl1_item2;
    GtkWidget *widgets_item;
    GtkWidget *label_widget, *button_widget, *checkbox_widget;
    GtkWidget *item2_item1, *item2_item1_item1;
    GtkWidget *label_item, *button_item, *checkbox_item;
    GtkWidget *cmd_cycle_line_style, *cmd_cycle_expander;

    GtkCTreeNode *root_node_1;
    GtkCTreeNode *root_node_2;
    GtkCTreeNode *root_node_3;

    /* Ponizej wpisujemy naglowki kolumn w obiekcie Ctree.
     * Szerokosc kolumny jest okreslona przez liczbe liter nazwy
     * naglowka, a nie przez dane, jakie zawiera.
     */

    gchar *column_titles[2] = {"Pozycja", "Informacja"};

    gchar *root1_data[2] = {"root1", "test"};
    gchar *root2_data[2] = {"root2", "Ten posiada wezly podrzedne"};
    gchar *root3_data[2] = {"Podwezeli1", "Podwezeli."};

    gtk_init(&argc, &argv);

    frm_trees = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title(GTK_WINDOW(frm_trees), "Prezentacja widgetow drzew");

    gtk_signal_connect(GTK_OBJECT(frm_trees),
        "destroy",
        GTK_SIGNAL_FUNC(destroy_main),
        NULL);

    hbox_main = gtk_hbox_new(TRUE, 0);
    vbox_left = gtk_vbox_new(TRUE, 0);
    vbox_right = gtk_vbox_new(FALSE, 0);

    /* Glowna roznica pomiedzy GtkTree a GtkCTree
     * jest (tak samo, jak miedzy GtkList a GtkCList) fakt, ze
     * GtkTree akceptuje wszystkie widgety,

```

```

    * a GtkCTree - tylko mapy pikselowe i tekst.
    */

tree_left = gtk_tree_new();

/* Drzewo nie jest przydatnym obiektem, jesli nic nie zawiera.
   * Na szczescie dodawanie nowych elementow do drzewa jest bardzo proste...
   */

lv11_item1 = gtk_tree_item_new_with_label("Poziom 1, Element 1");
lv11_item2 = gtk_tree_item_new_with_label("Poziom 1, Element 2");

gtk_tree_append(GTK_TREE(tree_left), lv11_item1);
gtk_tree_append(GTK_TREE(tree_left), lv11_item2);

/* Na razie nasze drzewo przypomina bardziej pole listy.
   * Zalety drzewa beda bardziej widoczne, gdy umiescimy w nim
   * poddrzewo. Poddrzewa dodaje sie troche inaczej od wezlow
   * i wskaznikow, ktore zapewniaja strukture poddrzewa.
   *
   * Nie bedziemy umieszczac "drzewa w drzewie", lecz dodamy
   * drzewo do elementu drzewa.
   */

subtree_item2 = gtk_tree_new();
item2_item1 = gtk_tree_item_new_with_label("Poziom 2, Element 1");
gtk_tree_append(GTK_TREE(subtree_item2), item2_item1);

gtk_tree_item_set_subtree(GTK_TREE_ITEM(lv11_item2), subtree_item2);

/* Teraz trzeba jawnie pokazac elementy poddrzewa.
   *
   * Jesli uzytkownik zaznaczy elementy poddrzewa, a nastepnie zwinie
   * jego galaz, to te elementy nadal beda zaznaczone.
   */

gtk_widget_show(item2_item1);

subtree_item3 = gtk_tree_new();
item2_item1_item1 = gtk_tree_item_new_with_label("Poziom 3, Element 1");
gtk_tree_append(GTK_TREE(subtree_item3), item2_item1_item1);

gtk_tree_item_set_subtree(GTK_TREE_ITEM(item2_item1), subtree_item3);

gtk_widget_show(item2_item1_item1);

widgets_item = gtk_tree_item_new_with_label("Widgety");
gtk_tree_append(GTK_TREE(tree_left), widgets_item);

subtree_widgets = gtk_tree_new();
gtk_tree_item_set_subtree(GTK_TREE_ITEM(widgets_item), subtree_widgets);

/* Dodajmy kilka widgetow do drzewa.
   *
   * Najpierw utworzymy etykiety:
   *
   */

```

```

label_item = gtk_tree_item_new();
label_widget = gtk_label_new("To jest etykieta");
gtk_container_add(GTK_CONTAINER(label_item), label_widget);
gtk_tree_append(GTK_TREE(subtree_widgets), label_item);
gtk_widget_show(label_widget);
gtk_widget_show(label_item);

/* Następnie umiemy w drzewie przycisk.
*/

button_item = gtk_tree_item_new();
button_widget = gtk_button_new_with_label("To jest przycisk.");
gtk_container_add(GTK_CONTAINER(button_item), button_widget);
gtk_tree_append(GTK_TREE(subtree_widgets), button_item);
gtk_widget_show(button_widget);
gtk_widget_show(button_item);

/* A teraz pole wyboru.
*/

checkbox_item = gtk_tree_item_new();
checkbox_widget = gtk_check_button_new_with_label("To jest pole wyboru.");
gtk_container_add(GTK_CONTAINER(checkbox_item), checkbox_widget);
gtk_tree_append(GTK_TREE(subtree_widgets), checkbox_item);
gtk_widget_show(checkbox_widget);
gtk_widget_show(checkbox_item);

/* Porównajmy to z drzewem CTree.
*/

ctree_right = gtk_ctree_new_with_titles(2, 0, column_titles);

root_node_1 = gtk_ctree_insert_node(GTK_CTREE(ctree_right), NULL, NULL, root1_data,
                                     0, NULL, NULL, NULL, NULL, FALSE, TRUE);
root_node_2 = gtk_ctree_insert_node(GTK_CTREE(ctree_right), NULL, NULL, root2_data,
                                     0, NULL, NULL, NULL, NULL, FALSE, TRUE);
root_node_3 = gtk_ctree_insert_node(GTK_CTREE(ctree_right), root_node_2, NULL,
                                     root3_data,
                                     0, NULL, NULL, NULL, NULL, FALSE, TRUE);

/* Konstrukcja drzewa CTree jest znacznie prostsza.
* Wystarczy utworzyć jeden egzemplarz tego widgetu, a następnie
* tworzyć węzły i łączyć je z węzłami w drzewie. Oto
* parametry dla funkcji gtk_ctree_insert_node:
*
*      ctree          Obiekt CTree.
*      parent         Oznacza węzeł nadrzędny, NULL dla węzła głównego.
*      sibling         Oznacza węzeł "siostrzany"; NULL to zakończenie galezi.
*      text            Tablica z wartościami poszczególnych kolumn;
*                     liczba wartości jest równa liczbie kolumn
*                     umieszczonych w ctree.
*      spacing        Liczba pikseli pomiędzy strukturą drzewa
*                     a tekstem.
*      pixmap_closed   \
*      bitmask_closed  \ Odpowiednie ikony, wyświetlane,
*      pixmap_opened   /   gdy węzeł jest rozwinięty lub zwinięty.
*      bitmask_opened  /
*      is_leaf        Wartość logiczna określająca, czy element nie jest węzłem;

```

```

*                                     TRUE oznacza, że element nie zawiera węzłów podrzędnych
*                                     (nie można tam umieścić żadnych węzłów podrzędnych).
*   expanded                         Wartość logiczna określająca, czy dany węzeł
*                                     jest rozwinięty czy nie.
*
*/

cmd_cycle_line_style = gtk_button_new_with_label("Style linii");
gtk_signal_connect(GTK_OBJECT(cmd_cycle_line_style),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_cycle_line_style_clicked),
    NULL);

cmd_cycle_expander = gtk_button_new_with_label("Style rozwijaczy");
gtk_signal_connect(GTK_OBJECT(cmd_cycle_expander),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_cycle_expander_clicked),
    NULL);

gtk_box_pack_start(GTK_BOX(vbox_left), tree_left, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(vbox_right), ctree_right, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(vbox_right), cmd_cycle_line_style, TRUE, FALSE, 0);
gtk_box_pack_start(GTK_BOX(vbox_right), cmd_cycle_expander, TRUE, FALSE, 0);

gtk_box_pack_start(GTK_BOX(hbox_main), vbox_left, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(hbox_main), vbox_right, TRUE, TRUE, 0);
gtk_container_add(GTK_CONTAINER(frm_trees), hbox_main);

gtk_widget_show_all (frm_trees);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

void cmd_cycle_line_style_clicked()
{
    static gint line_style = 1;

    if (line_style == 4)
        line_style = 1;
    else line_style++;

    switch (line_style)
    {
        case 1: gtk_ctree_set_line_style(GTK_CTREE(ctree_right),
                                         GTK_CTREE_LINES_NONE);
                break;
        case 2: gtk_ctree_set_line_style(GTK_CTREE(ctree_right),
                                         GTK_CTREE_LINES_SOLID);
                break;
        case 3: gtk_ctree_set_line_style(GTK_CTREE(ctree_right),
                                         GTK_CTREE_LINES_DOTTED);
                break;
    }
}

```

```
        case 4: gtk_ctree_set_line_style(GTK_CTREE(ctree_right),
                                         GTK_CTREE_LINES_TABBED);
                break;
        default: g_print("Ta linia nie powinna byc widoczna!!");
                break;
    }
}

void cmd_cycle_expander_clicked()
{
    static gint expander_style = 1;

    if (expander_style == 4)
        expander_style = 1;
    else expander_style++;

    switch (expander_style)
    {
        case 1: gtk_ctree_set_expander_style(GTK_CTREE(ctree_right),
                                              GTK_CTREE_EXPANDER_NONE);
                break;
        case 2: gtk_ctree_set_expander_style(GTK_CTREE(ctree_right),
                                              GTK_CTREE_EXPANDER_SQUARE);
                break;
        case 3: gtk_ctree_set_expander_style(GTK_CTREE(ctree_right),
                                              GTK_CTREE_EXPANDER_TRIANGLE);
                break;
        case 4: gtk_ctree_set_expander_style(GTK_CTREE(ctree_right),
                                              GTK_CTREE_EXPANDER_CIRCULAR);
                break;
        default: g_print("Ta linia nie powinna byc widoczna!!");
                break;
    }
}
```

GtkFixed

Obiekt `GtkFixed` jest obiektem docenianym szczególnie przez programistów wykorzystujących wcześniej VB lub VC++ ze względu na jedną, ważną właściwość. Otóż w tym obiekcie możemy umieszczać inne widgety bezpośrednio na jego powierzchni (pokrytej siatką, ułatwiającą lokalizację rozmieszczanych widжетów — rysunek 4.3). Nie korzystamy tu z obiektów-pojemników, których stosowanie jest główną własnością GTK+. Większość programistów zamiast tego obiektu wybierze `GtkLayout`, ze względu na jego większe możliwości (obiekt zaprezentowano w kolejnej sekcji, „`GtkLayout`”).

Program umieszczony w listingu 4.3 jest bardzo prosty. Wykorzystano w nim obiekt `GtkFixed` i jeden przycisk, który po każdym kliknięciu zmienia swoją pozycję na powierzchni okna `GtkFixed`. Porównajmy `GtkFixed` z obiektem `GtkLayout` przed podjęciem decyzji, który z nich lepiej nadaje się do wykorzystania w naszych aplikacjach.

Rysunek 4.3.
Obiekt GtkFixed



Listing 4.3. *Prezentacja GtkFixed*

```
#include <gtk/gtk.h>
#include <stdlib.h>
/* Biblioteka <stdlib.h> jest potrzebna do
 * wyznaczania wartosci losowych w funkcji zwrotnej. Wartosci
 * losowe beda potrzebna do przemieszczania przycisku po
 * powierzchni obiektu GtkFixed.
 */

GtkWidget *frm_fixed;
GtkWidget *fixed_main;
GtkWidget *cmd1;

void destroy_main();
void cmd1_clicked();

gint main(gint argc, gchar *argv[])
{
    gtk_init(&argc, &argv);

    frm_fixed = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title(GTK_WINDOW(frm_fixed), "Prezentacja obiektu GtkFixed");

    gtk_signal_connect(GTK_OBJECT(frm_fixed),
        "destroy",
        GTK_SIGNAL_FUNC(destroy_main),
        NULL);

    fixed_main = gtk_fixed_new();

    /* GtkFixed jest obiektem prostym w uzyciu. Nalezy
     * podac wspolrzedne polozenia kazdego widgetu potomnego,
     * umieszczanego na powierzchni GtkFixed. Atrybuty fill,
     * expand, padding itd. sa tu bezuzyteczne; obiekty
     * potomne, umieszczone na powierzchni GtkFixed, beda domyslnie
     * zajmowac najmniejszy obszar, pozwalajacy na ich wyswietlenie.
     * Tak samo zachowa sie sam obiekt GtkFixed, ktory bedzie domyslnie
     * zajmowal minimalna powierzchnie ekranu, potrzebna do wyswietlenia
     * wszystkich jego obiektow. Jesli widget potomny zostanie przesuniety
     * poza biezacy obszar GtkFixed, to widget ten zmieni odpowiednio
     * swoje wymiary tak, aby przesuniety obiekt nadal byl widoczny.
     * Jesli obiekt potomny zostanie pozniej umieszczony w srodku
     * okna glownego, GtkFixed nie zmniejszy automatycznie
     * swej powierzchni.
     *
     *
     * Moze to prowadzic do dziwnych i nieoczekiwanych zachowan
```

```

* okna glownego w trakcie dzialania programu. Zalozy
* przykladowa pozycje obiektu 1,1. Przesunmy obiekt na pozycje 100, 100
* (okno glowne dostosuje sie do nowego polozenia). Gdy zmienimy polozenie
* obiektu na 50,50, okno nie zmniejszy sie. Gdy jednak przeniesiemy
* obiekt na 150,50, to okno glowne znów zmieni swój rozmiar tak, aby
* wyświetlany obiekt zmieścił się na jego powierzchni.
* Okno glowne jest rysowane ponownie za każdym razem, gdy obiekty należące do
* niego zostaną umieszczone poza powierzchnią GtkFixed. Okno glowne, które
* pierwotnie było kwadratem, może stać się w pewnych przypadkach np.
* bardzo cienkim i długim prostokątem.
*
* Choć GtkFixed jest bardzo łatwy w zastosowaniu,
* programiści częściej korzystają z GtkLayout.
* Przed dokonaniem wyboru "ulubionego okna" sprawdźmy
* wady i zalety każdego z nich (GtkFixed i GtkLayout).
*/

cmd1 = gtk_button_new_with_label("Przycisk 1");
gtk_signal_connect(GTK_OBJECT(cmd1),
    "clicked",
    GTK_SIGNAL_FUNC(cmd1_clicked),
    NULL);
gtk_fixed_put(GTK_FIXED(fixed_main), cmd1, 1, 100);

/* Trzecim i czwartym parametrem powyższej funkcji
* są odpowiednio: odległość od lewej krawędzi okna (w pikselach)
* i odległość od górnej krawędzi okna (również
* w pikselach).
*/

gtk_container_add(GTK_CONTAINER(frm_fixed), fixed_main);

gtk_widget_show_all (frm_fixed);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

void cmd1_clicked()
{
    gint x_pos, y_pos;

    x_pos = rand();
    y_pos = rand();

    /* Tu generujemy liczby losowe w zakresie od 1 do 200,
    * które posłużą nam jako nowe lokalizacje przycisku.
    */

    do{
        x_pos = x_pos - 200;
    } while (x_pos > 200);

```

```

do{
    y_pos = y_pos - 200;
} while (y_pos > 200);

gtk_fixed_move(GTK_FIXED(fixed_main), GTK_WIDGET(cmd1), x_pos, y_pos);
}

```

GtkLayout

GtkLayout ma podobne właściwości, co GtkFixed, ale umożliwia wykorzystanie większej powierzchni dla widgetów potomnych od powierzchni wymaganej do ich wyświetlenia. W tym obiekcie, zamiast korzystać z niszczenia i ponownego tworzenia okna, mamy do dyspozycji funkcje *_freeze_* i *_thaw_* (opisane w listingu 4.4). Na rysunku 4.4 przedstawiono wynik działania programu.

Rysunek 4.4.
*Prezentacja działania
GtkLayout*



Aby uwidocznic podobieństwa i różnice pomiędzy kontrolkami GtkFixed i GtkLayout, listing 4.4 jest bardzo podobny do listingu 4.3. Najlepiej oba programy skompilować i uruchomić, a następnie wybrać bardziej odpowiedni dla danego zastosowania obiekt.

Listing 4.4. *Prezentacja GtkLayout*

```

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

GtkWidget *frm_layout;
GtkWidget *layout_main;
GtkWidget *cmd1;

void destroy_main();
void cmd1_clicked();

gint main(gint argc, gchar *argv[])
{
    gtk_init(&argc, &argv);

    frm_layout = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title(GTK_WINDOW(frm_layout), "Prezentacja obiektu GtkLayout");

    gtk_signal_connect(GTK_OBJECT(frm_layout),
        "destroy",
        GTK_SIGNAL_FUNC(destroy_main),
        NULL);

```

```

layout_main = gtk_layout_new(NULL, NULL);

/* Zamiast wywoływać funkcję gtk_layout_new(), która jako parametrow
 * potrzebuje obiektów regulacyjnych (poziomych i pionowych),
 * można skorzystać z poprzedniego listingu
 * i w instrukcjach tworzących obiekty zamienić
 * słowo "fixed" na "layout" (program powinien zadziałać).
 *
 * Ten program jest podobny do poprzedniego, ale posiada
 * kilka dodatkowych instrukcji, prezentujących
 * możliwości obiektu GtkLayout.
 */

cmd1 = gtk_button_new_with_label("Przycisk 1");
gtk_signal_connect(GTK_OBJECT(cmd1),
                  "clicked",
                  GTK_SIGNAL_FUNC(cmd1_clicked),
                  NULL);
gtk_layout_put(GTK_LAYOUT(layout_main), cmd1, 1, 100);

gtk_container_add(GTK_CONTAINER(frm_layout), layout_main);

gtk_widget_show_all (frm_layout);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

void cmd1_clicked()
{
    gint x_pos, y_pos;

    x_pos = rand();
    y_pos = rand();

    /* Tu generujemy liczby losowe w zakresie od 1 do 200,
     * które posłużą nam jako nowe lokalizacje przycisku.
     */

    do{
        x_pos = x_pos - 200;
    } while (x_pos > 200);

    do{
        y_pos = y_pos - 200;
    } while (y_pos > 200);

    /* Oto największa różnica pomiędzy GtkFixed a GtkLayout.
     * Zamiast rysować od nowa okno wraz z jego elementami, lepiej
     * zablokować je, a po uprzednim przemieszczeniu jego obiektów -
     * odblokować. Jest to szczególnie przydatne przy wielu obiektach
     * potomnych.
     */

```

```
gtk_layout_freeze(GTK_LAYOUT(layout_main));

gtk_layout_move(GTK_LAYOUT(layout_main), GTK_WIDGET(cmd1), x_pos, y_pos);

gtk_layout_thaw(GTK_LAYOUT(layout_main));
}
```

GtkScrolledWindow

Obiekt `GtkScrolledWindow` jest pojemnikiem, w którym można umieścić inne widgety. Okno to posiada paski przewijania. Można je zaimplementować również w innych widgetach, które nie posiadają tych pasków. Dzięki temu na jednym ekranie można umieścić kilka okienek, w których — za pomocą pasków przewijania — można wybierać wyświetlany obszar z pożądanymi obiektami. Na rysunku 4.5 przedstawiono okno wygenerowane przez program, którego kod źródłowy znajduje się w listingu 4.5.

Rysunek 4.5.

Obiekt

GtkScrolledWindow



W programie prezentującym okno przewijane umieściliśmy dwa przyciski, dzięki którym możemy wybrać odpowiednie ustawienia. Jeden z nich służy do wybrania opcji *always*, czyli stałego wyświetlania pasków przewijania (niezależnie od tego, czy jest to potrzebne, czy nie), a drugi — do ustawienia opcji *auto*, czyli wyświetlania pasków tylko wtedy, gdy widget w oknie głównym nie mieści się na wyświetlanym obszarze. Należy wiedzieć, że odpowiednich ustawień można dokonać dla każdego paska osobno. Na przykład dla paska pionowego można wybrać właściwość *auto*, a dla poziomego — *always*. W naszym programie będziemy wybierać takie same właściwości dla obydwu pasków.

Listing 4.5. Prezentacja *GtkScrolledDemo*

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

GtkWidget *frm_scrolled;
GtkWidget *scrolled_main;
GtkWidget *fixed_main;
GtkWidget *cmd_set_policy_auto, *cmd_set_policy_always;

void destroy_main();
void cmd_set_policy_auto_clicked();
void cmd_set_policy_always_clicked();
```

```
gint main(gint argc, gchar *argv[])
{
    gtk_init(&argc, &argv);

    frm_scrolled = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(frm_scrolled), "Prezentacja okna z paskami
    przewijania");

    /* Należy zauważyć, że nie podając początkowych rozmiarów
    * okna zgadzamy się na to, aby okno zajęło bardzo małą
    * powierzchnię. Aby wyświetlić obydwa przyciski, trzeba
    * będzie zmienić rozmiary okna głównego. Pokażemy tu również
    * dwa sposoby wyświetlania pasków przewijania.
    */

    gtk_signal_connect(GTK_OBJECT(frm_scrolled),
        "destroy",
        GTK_SIGNAL_FUNC(destroy_main),
        NULL);

    scrolled_main = gtk_scrolled_window_new(NULL, NULL);
    fixed_main = gtk_fixed_new();

    cmd_set_policy_auto = gtk_button_new_with_label("Paski wyświetlane w razie
    potrzeby");
    gtk_signal_connect(GTK_OBJECT(cmd_set_policy_auto),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_set_policy_auto_clicked),
        NULL);

    cmd_set_policy_always = gtk_button_new_with_label("Paski zawsze widoczne");
    gtk_signal_connect(GTK_OBJECT(cmd_set_policy_always),
        "clicked",
        GTK_SIGNAL_FUNC(cmd_set_policy_always_clicked),
        NULL);

    gtk_fixed_put(GTK_FIXED(fixed_main), cmd_set_policy_auto, 200, 200);
    gtk_fixed_put(GTK_FIXED(fixed_main), cmd_set_policy_always, 50, 75);

    gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrolled_main),
        fixed_main);
    gtk_container_add(GTK_CONTAINER(frm_scrolled), scrolled_main);

    gtk_widget_show_all (frm_scrolled);
    gtk_main ();
    return 0;
}

void destroy_main()
{
    gtk_main_quit();
}

void cmd_set_policy_auto_clicked()
{
    gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled_main),
        GTK_POLICY_AUTOMATIC,
        GTK_POLICY_AUTOMATIC);
}
```

```

    }

    void cmd_set_policy_always_clicked()
    {
        /* To jest domyslnie zachowanie sie widgetu.
        */

        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled_main),
                                       GTK_POLICY_ALWAYS,
                                       GTK_POLICY_ALWAYS);
    }

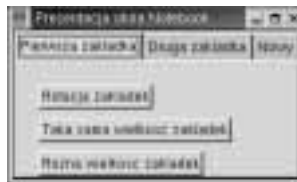
```

GtkNotebook

Obiekt `GtkNotebook` jest odpowiednikiem kontrolki nazywanej przez programistów VB „kontrolką zakładek”. Taka kontrolka umożliwia egzystencję wielu obiektów na jednej „powierzchni użytkowej”. Każda zakładka oznacza osobną stronę, na której można umieścić dowolne obiekty kontrolne. W danej chwili można wyświetlić jedną stronę, a inne w tym czasie pozostaną ukryte.

Okno z główną zakładką aplikacji pokazano na rysunku 4.6, a odpowiedni kod źródłowy w listingu 4.6. Zwróćmy uwagę na możliwość dynamicznego dodawania kolejnych stron (w zasadzie można napisać program, w którym można dodawać również inne widgety w trakcie jego działania).

Rysunek 4.6.
Uruchomiony program
prezentujący
działanie
`GtkNotebook`



Listing 4.6. Program prezentujący obiekt `GtkPaned`

```

#include <gtk/gtk.h>

GtkWidget *frm_notebook;
GtkWidget *notebook_main;
GtkWidget *fixed_page1;
GtkWidget *vbox_page2;

void destroy_main();
void cmd_rotate_tabs_clicked();
void cmd_set_tabs_homogeneous_clicked();
void cmd_set_tabs_heterogeneous_clicked();
void cmd_add_page_clicked();
void cmd_get_page_index_clicked();
void notebook_switch_page();

gint main(gint argc, gchar *argv[])
{
    GtkWidget *cmd_rotate_tabs;

```

```
GtkWidget *cmd_set_tabs_homogeneous, *cmd_set_tabs_heterogeneous;
GtkWidget *cmd_add_page, *cmd_get_page_index;
GtkWidget *lbl_tab1, *lbl_tab2;

gtk_init(&argc, &argv);

frm_notebook = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(frm_notebook), "Prezentacja okna Notebook");

gtk_signal_connect(GTK_OBJECT(frm_notebook),
    "destroy",
    GTK_SIGNAL_FUNC(destroy_main),
    NULL);

notebook_main = gtk_notebook_new();
gtk_signal_connect(GTK_OBJECT(notebook_main),
    "switch-page",
    GTK_SIGNAL_FUNC(notebook_switch_page),
    NULL);

/* Strona pierwsza w notatniku...
*/

fixed_page1 = gtk_fixed_new();
lbl_tab1 = gtk_label_new("Pierwsza zakladka");

cmd_rotate_tabs = gtk_button_new_with_label("Rotacja zakladek");
gtk_signal_connect(GTK_OBJECT(cmd_rotate_tabs),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_rotate_tabs_clicked),
    NULL);

cmd_set_tabs_homogeneous = gtk_button_new_with_label("Taka sama wielkosc zakladek");
gtk_signal_connect(GTK_OBJECT(cmd_set_tabs_homogeneous),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_set_tabs_homogeneous_clicked),
    NULL);

cmd_set_tabs_heterogeneous = gtk_button_new_with_label("Rozna wielkosc zakladek");
gtk_signal_connect(GTK_OBJECT(cmd_set_tabs_heterogeneous),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_set_tabs_heterogeneous_clicked),
    NULL);

gtk_fixed_put(GTK_FIXED(fixed_page1), cmd_rotate_tabs, 20, 20);
gtk_fixed_put(GTK_FIXED(fixed_page1), cmd_set_tabs_homogeneous, 20, 50);
gtk_fixed_put(GTK_FIXED(fixed_page1), cmd_set_tabs_heterogeneous, 20, 80);
gtk_notebook_append_page(GTK_NOTEBOOK(notebook_main), fixed_page1, lbl_tab1);

/* Strona 2...
*/

vbox_page2 = gtk_vbox_new(TRUE, 0);
cmd_add_page = gtk_button_new_with_label("Dodaj strone");
gtk_signal_connect(GTK_OBJECT(cmd_add_page),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_add_page_clicked),
    NULL);
```

```

cmd_get_page_index = gtk_button_new_with_label("Pobierz indeks strony");
gtk_signal_connect(GTK_OBJECT(cmd_get_page_index),
    "clicked",
    GTK_SIGNAL_FUNC(cmd_get_page_index_clicked),
    NULL);

gtk_box_pack_start_defaults(GTK_BOX(vbox_page2), cmd_add_page);
gtk_box_pack_start_defaults(GTK_BOX(vbox_page2), cmd_get_page_index);

lbl_tab2 = gtk_label_new("Druga zakładka");

gtk_notebook_append_page(GTK_NOTEBOOK(notebook_main), vbox_page2, lbl_tab2);

/* Koniec drugiej zakładki, czas na złożenie okna głównego.
*/

gtk_container_add(GTK_CONTAINER(frm_notebook), notebook_main);

gtk_widget_show_all (frm_notebook);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

void cmd_rotate_tabs_clicked()
{
    static gint tab_location = 1;

    if (tab_location == 4)
        tab_location = 1;
    else tab_location++;

    switch (tab_location)
    {
        case 1: gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook_main), GTK_POS_TOP);
                break;
        case 2: gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook_main), GTK_POS_RIGHT);
                break;
        case 3: gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook_main), GTK_POS_BOTTOM);
                break;
        case 4: gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook_main), GTK_POS_LEFT);
                break;
        default: g_print("Ten przypadek nie powinien nigdy wystąpić.\n");
    }
}

void cmd_set_tabs_homogeneous_clicked()
{
    gtk_notebook_set_homogeneous_tabs(GTK_NOTEBOOK(notebook_main), TRUE);
}

void cmd_set_tabs_heterogeneous_clicked()

```

```
{
    /* Ta funkcja nie ma efektu, jeśli zakładki umieszczone są po prawej lub lewej
    * stronie. W tym przypadku funkcja zachowa się tak samo, jak
    * cmd_set_tabs_homogeneous_clicked().
    */
    gtk_notebook_set_homogeneous_tabs(GTK_NOTEBOOK(notebook_main), FALSE);
}

void cmd_add_page_clicked()
{
    GtkWidget *fixed;
    GtkWidget *lbl;

    fixed = gtk_fixed_new();

    lbl = gtk_label_new("Nowy");

    gtk_notebook_append_page(GTK_NOTEBOOK(notebook_main), fixed, lbl);

    /* Odświeżenie okna.
    */

    gtk_widget_show_all(notebook_main);
}

void cmd_get_page_index_clicked()
{
    /* Zakładki są numerowane od zera. */
    g_print("Indeks strony to %i.\n",
    gtk_notebook_get_current_page(GTK_NOTEBOOK(notebook_main)));
}

void notebook_switch_page()
{
    g_print("stronę zmieniono.\n");
}
```

GtkPaned

Dzięki widgetowi `GtkPaned` można podzielić okno — pionowo lub poziomo — na dwie mniejsze powierzchnie, zwane panelami (rysunek 4.7). Jeśli chcemy uzyskać więcej mniejszych powierzchni, możemy dzielić większe na dwie mniejsze, potem mniejsze na jeszcze mniejsze, i tak dalej. Po uruchomieniu programu w oknie zobaczymy specjalne przyciski, dzięki którym będziemy mogli zmienić rozmiary każdej powierzchni (pod warunkiem, że wybierzemy właściwe opcje).

Zwróćmy uwagę na wymiary przycisków służących do zmiany rozmiarów powierzchni. W naszym przypadku mamy do czynienia z bardzo „cieniutkimi” obiektami, ale nic nie stoi na przeszkodzie, aby wykorzystać większe przyciski. Wymiary tych przycisków są związane z przestrzenią pomiędzy granicą powierzchni a obiektami w niej umieszczonymi. Wielkość tej przestrzeni określa się w funkcji `gtk_paned_gutter_size()`.

Rysunek 4.7.
Okno prezentacji
GtkPaned



Listing 4.7. Prezentacja GtkPaned

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

GtkWidget *frm_paned;

void destroy_main();

gint main(gint argc, gchar *argv[])
{
    GtkWidget *lbl_hpaned1_left;
    GtkWidget *lbl_vpaned1_top;
    GtkWidget *lbl_hpaned2_left;
    GtkWidget *lbl_vpaned2_top;
    GtkWidget *lbl_vpaned2_bottom;
    GtkWidget *hpaned1;
    GtkWidget *vpaned1;
    GtkWidget *hpaned2;
    GtkWidget *vpaned2;

    gtk_init(&argc, &argv);

    frm_paned = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(frm_paned), "Prezentacja GtkPaned");

    gtk_signal_connect(GTK_OBJECT(frm_paned),
        "destroy",
        GTK_SIGNAL_FUNC(destroy_main),
        NULL);

    hpaned1 = gtk_hpaned_new();
    gtk_paned_gutter_size(GTK_PANED(hpaned1), 20);
    gtk_paned_handle_size(GTK_PANED(hpaned1), 20);
    lbl_hpaned1_left = gtk_label_new("hpaned1 na lewo\nOpcja Resize TRUE\nOpcja Shrink TRUE");
    gtk_paned_pack1(GTK_PANED(hpaned1), lbl_hpaned1_left, TRUE, TRUE);

    /* Wywołanie funkcji gtk_paned_gutter_size() umożliwia ustalenie odległości
     * pomiędzy panelem a widgetami istniejącymi obok. Warto zauważyć,
     * że niewielki odstęp wygląda estetycznie w interfejsie użytkownika.
     *
     * gtk_paned_handle_size() powoduje umieszczenie małego, kwadratowego przycisku,
     * dzięki któremu użytkownik może zmienić rozmiary panelu.
     *
     * gtk_paned_pack1() powoduje umieszczenie innego obiektu
     * w lewej, poziomej części panelu lub w górnej,
     * pionowej części panelu. gtk_paned_pack2() służy (patrz poniżej)
     * do umieszczania obiektów w dolnej lub prawej części
    */
}
```

```

* panelu hpaned lub vpaned.
*
* Parametry funkcji gtk_paned_pack*():
*   pane    panel docelowy;
*   child   widget potomny, umieszczany w lewej (prawej)
*           lub górnej (dolnej) części;
*   resize   jeśli ma wartość TRUE, obiekt potomny zmieni rozmiary odpowiednio
*           do zmian proporcji w oknie głównym;
*   shrink   jeśli ma wartość FALSE, panel nie będzie mógł mieć powierzchni
*           mniejszej niż powierzchnia obiektu potomnego.
*/

vpaned1 = gtk_vpaned_new();
gtk_paned_gutter_size(GTK_PANED(vpaned1), 20);
lbl_vpaned1_top = gtk_label_new("vpaned1 góra\nOpcja Resize TRUE\nOpcja Shrink
    FALSE");
gtk_paned_pack2(GTK_PANED(hpaned1), vpaned1, TRUE, FALSE);
gtk_paned_pack1(GTK_PANED(vpaned1), lbl_vpaned1_top, TRUE, FALSE);

hpaned2 = gtk_hpaned_new();
gtk_paned_handle_size(GTK_PANED(hpaned2), 20);
lbl_hpaned2_left = gtk_label_new("hpaned2 na lewo\nOpcja Resize FALSE\nOpcja Shrink
    TRUE");
gtk_paned_pack2(GTK_PANED(vpaned1), hpaned2, FALSE, TRUE);
gtk_paned_pack1(GTK_PANED(hpaned2), lbl_hpaned2_left, FALSE, TRUE);

vpaned2 = gtk_vpaned_new();
gtk_paned_gutter_size(GTK_PANED(vpaned2), 5);
gtk_paned_handle_size(GTK_PANED(vpaned2), 5);
lbl_vpaned2_top = gtk_label_new("vpaned2 góra\nResize FALSE\nOpcja Shrink FALSE");
lbl_vpaned2_bottom = gtk_label_new("vpaned2 dół");
gtk_paned_pack2(GTK_PANED(hpaned2), vpaned2, FALSE, FALSE);
gtk_paned_pack1(GTK_PANED(vpaned2), lbl_vpaned2_top, FALSE, FALSE);
gtk_paned_pack2(GTK_PANED(vpaned2), lbl_vpaned2_bottom, FALSE, FALSE);

gtk_container_add(GTK_CONTAINER(frm_paned), hpaned1);

gtk_widget_show_all (frm_paned);
gtk_main ();
return 0;

}

void destroy_main()
{
    gtk_main_quit();
}

```
