Rozdział 11

Style, kolory, czcionki, wskaźniki myszy i referencje

Style mają wpływ na wygląd rysowanej kontrolki, zwłaszcza na jej kolor i używaną czcionkę. Każdy styl składa się z kilku kolorów i jednej czcionki. Style są częścią GTK+, ale kolory i czcionki są częścią GDK. Chociaż obsługa kolorów jest zawarta w GDK, GTK+ posiada okno wyboru koloru, które można wykorzystać w swoich aplikacjach. Jeśli program tego wymaga, GTK+ dysponuje także oknem wyboru czcionek. Oba te okna dialogowe przypominają okno wyboru pliku i ujednolicają interfejs użytkownika w tych aplikacjach GTK+, które umożliwiają użytkownikom wybieranie koloru i czcionki. GDK pozwala także na zmianę wskaźnika myszy na jeden z wbudowanych kształtów albo na kształt zdefiniowany przez programistę. Referencje pomagają GTK+ w obsłudze używanych kontrolek. GTK+ zwalnia kontrolki, kiedy sądzi, że nie są już potrzebne, czemu w pewnych sytuacjach należy zapobiegać.

Style (kolory i czcionki)

Każda kontrolka w GTK+ posiada styl, który określa jej wygląd na ekranie. Podstawowy styl składa się ze schematu kolorów i czcionki. Aplikacje GTK+ zazwyczaj dzielą typowy styl i właśnie dlatego wszystkie kontrolki mają te same kolory i korzystają z tej samej czcionki. Kontrolka może posiadać swój własny, niepowtarzalny styl, ale definiowanie odrębnego stylu dla każdej kontrolki jest raczej nie najlepszym pomysłem. Wyświetlanie wielu kontrolek o odmiennych stylach odciąga uwagę od samego programu, zwłaszcza wtedy, kiedy kontrolki mają różne kolory. Lepiej jest ograniczyć liczbę styli do najwyżej kilku (albo jednego), a zaoszczędzony czas przeznaczyć na zwiększenie funkcjonalności aplikacji.

Chociaż w stylu określona jest tylko jedna czcionka, to może on mieć wiele kolorów. Kontrolka może znajdować się w jednym z kilku stanów:

NORMAL, ACTIVE, PRELIGHT, SELECTED i INSENSITIVE. Każdemu stanowi można przypisać odrębny schemat kolorystyczny.

NORMAL	Zwykły sposób rysowania obiektu.
ACTIVE	Obiekt jest aktywny - w przypadku przycisku stan ACTIVE występuje wtedy, kiedy przycisk jest wciśnięty.
PRELIGHT	Zazwyczaj oznacza, że nad obiektem przesuwa się wskaźnik myszy. Kolor ten podpowiada użytkownikowi, że może kliknąć na obiekcie.
SELECTED	Obiekt został zaznaczony.
INSENSITIVE	Kontrolka jest niewrażliwa (nie można jej wybrać). W terminologii firmy Microsoft stan taki jest określany mianem nieaktywnego (disabled)

Każdy zdefiniowany zbiór kolorów wyraża stan, w którym znajduje się kontrolka. Można obejrzeć je w działaniu na przykładzie kontrolki przycisku. Przycisk zazwyczaj znajduje się w stanie NORMAL. Kiedy przesuniemy nad niego wskaźnik myszy, przycisk znajdzie się w stanie PRELIGHT i będzie wyświetlany w kolorach właściwych dla tego stanu. Jeśli klikniemy przycisk, zostanie on wyświetlony w kolorach zdefiniowanych dla stanu ACTIVE. Nie każda kontrolka może znajdować się we wszystkich stanach.

Kolory

Kolory definiujemy jako strukturę GdkColor. Kolor posiada trzy tworzące go składowe: czerwoną, zieloną i niebieską (RGB).

```
/* --- definiujemy kolor czerwony: dużo czerwonego, bez niebieskiego i
  --- zielonego */
GdkColor kolor = {0, 0xffff, 0x0000, 0x0000};
```

Po stworzeniu struktur GdkColor trzeba zażądać przydzielenia koloru z zasobów systemowych przy pomocy funkcji gdk_color_alloc. Funkcja ta próbuje dopasować zdefiniowany kolor do kolorów udostępnianych przez system. Funkcja przyjmuje mapę kolorów i kolor, ale jeśli korzystamy z niewielu kolorów, możemy przekazać jej systemową mapę kolorów, zwracaną przez funkcję gdk_colormap_get_system. Po przydzieleniu kolorów można wykorzystać je do tworzenia nowych, (oby) bardziej interesujących stylów.

Korzystanie ze stylów

Można pobrać domyślny styl aplikacji przy pomocy funkcji gtk_widget_get_default_style. Bezpośrednia ingerencja w ten styl jest złą praktyką, ale możemy stworzyć jego kopię. Funkcja gtk_style_copy tworzy kopię stylu, którą można poddać dowolnym manipulacjom. Struktura GtkStyle zawiera tablicę kolorów pierwszoplanowych (*foreground colors*, fg), kolorów tła (*background colors*, bg) i kolorów tekstu (*text colors*, text), która definiuje kolory stylu. Aby zmodyfikować przycisk tak, żeby wyświetlał biały tekst na czarnym tle, możemy:

```
/* --- zdefiniować kolory --- */
GdkColor bialy = {0, 0xffff, 0xffff, 0xffff);
GdkColor czarny = {0, 0x0000, 0x0000, 0x0000);

/* --- przydzielić kolory --- */
gdk_color_alloc (gdk_colormap_get_system (), &bialy);
gdk_color_alloc (gdk_colormap_get_system (), &czarny);

/* --- pobrać domyślny styl --- */
styl_domyslny = gtk_widget_get_default_style ();

/* --- wykonać kopię stylu na własny użytek --- */
styl = gtk_style_copy (styl_domyslny);

/* --- zmodyfikować kolory --- */
styl->fg[NORMAL] = bialy;
styl->text[NORMAL] = bialy;
styl->bg[NORMAL] = czarny;
```

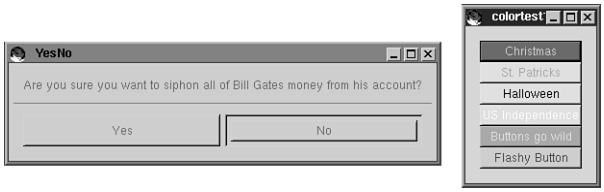
Po utworzeniu stylu można zmieniać z jego pomocą wygląd dowolnej kontrolki. Funkcja gtk_widget_set_style ustawia styl istniejącej kontrolki, ale nie zmienia stanu żadnych jej potomków. Można skorzystać z funkcji gtk_container_foreach, aby zmienić styl kontrolki i wszystkich zawartych w niej kontrolek, na przykład w taki sposób:

```
void UstawStylRekurencyjnie (GtkWidget *kontrolka, gpointer dane)
{
   GtkStyle *styl;

   /* --- pobieramy styl --- */
   styl = (GtkStyle *) dane;

   /* --- ustawiamy styl kontrolki --- */
   gtk_widget_set_style (kontrolka, styl);
```

```
    /* --- jeśli kontrolka może posiadać potomków --- */
    if (GTK_IS_CONTAINER (kontrolka)) {

      /* --- ustawiamy także styl wszystkich potomków --- */
      gtk_container_foreach (GTK_CONTAINER (kontrolka),
                  UstawStylRekurencyjnie, styl);
    }
}
```

Zamiast zmieniać styl kontrolek już po ich utworzeniu, można w łatwy sposób ustawić styl stosowany do nowo tworzonych kontrolek. Funkcja gtk_widget_push_style ustawia bieżący styl dla wszystkich tworzonych kontrolek. Każda kontrolka, utworzona po wywołaniu funkcji gtk_widget_push_style, otrzymuje styl odłożony na stos. Kiedy styl nie jest już potrzebny, należy zdjąć go ze stosu funkcją gtk_widget_pop_style. Ponieważ style są umieszczone na stosie, wiele stylów może być zdefiniowanych jednocześnie, ale używać możemy tylko stylu z wierzchołka stosu. Jeśli przy pomocy funkcji gtk_widget_pop_style zdejmiemy ze stosu najwyżej położony styl, wówczas kontrolki będą tworzone przy użyciu następnego w kolejności stylu. Kiedy styl kontrolki ulegnie zmianie, otrzyma ona sygnał "style_set".

Poniższy przykład pokazuje, w jaki sposób można zmieniać kolor przycisków za pomocą stylów i jak ustawiać domyślny styl nowo tworzonych kontrolek. Program wyświetla sześć przycisków - pierwsze cztery są umieszczone wewnątrz okna dialogowego o określonym schemacie kolorystycznym (wskazówka: schemat „Boże Narodzenie" jest zielono-czerwony). Piąty przycisk sprawia, że wszystkie przyciski zmieniają kolory. W szóstym przycisku zmodyfikowane są wszystkie style, więc od początku jest wyświetlany w innych kolorach; jeśli przesuniemy nad niego wskaźnik myszy, zobaczymy, że w stanie PRELIGHT jest żółty - inaczej, niż w domyślnym schemacie (po kliknięciu przycisku „Przyciski szaleją" początkowy styl zostanie zastąpiony innym). Nie umieszczamy tutaj kodu funkcji WywolajOkno, która jest włączona do innego pliku. Otwiera ona okno dialogowe i wyświetla komunikat (patrz rysunek 11.1). Można rzucić okiem na jej kod źródłowy, ale zasadnicza część programu znajduje się poniżej.



Rysunek 11.1. Okno dialogowe modyfikujące kolory i rezultaty modyfikacji.

```
/*
 * Plik: przycisk.c
 * Autor: Eric Harlow
 *
 * Pokazuje sposób modyfikowania stylów - zwłaszcza kolorów
 *
 * --- NORMAL, PRELIGHT, ACTIVE, INSENSITIVE, SELECTED
 */

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

GtkWidget *przycisk;
GtkStyle *stylCzerwony;

/*
 * --- Definicje kolorów
 */
GdkColor czerwony = {0, 0xffff, 0x0000, 0x0000};
GdkColor niebieski = {0, 0x0000, 0x0000, 0xffff};
GdkColor zielony = {0, 0x0000, 0xffff, 0x0000};
GdkColor zolty = {0, 0xffff, 0xffff, 0x0000};
GdkColor fioletowy = {0, 0xffff, 0x0000, 0xffff};
GdkColor pomaranczowy = {0, 0xffff, 0x9999, 0x0000};
GdkColor turkusowy = {0, 0x0000, 0xffff, 0xffff};
GdkColor czarny = {0, 0x0000, 0x0000, 0x0000};
GdkColor bialy = {0, 0xffff, 0xffff, 0xffff};

/* --- Tworzymy listę kolorów, z której będzie korzystał
 *     przycisk przypisujący losowe kolory
 */
GdkColor listakolorow[] = {
   {0,      0xffff,     0x0000,    0x0000},
   {0,      0x0000,     0x0000,    0xffff},
```

```
    {0,     0x0000,   0xffff,     0x0000},
    {0,     0xffff,   0xffff,     0x0000},
    {0,     0xffff,   0x0000,     0xffff},
    {0,     0xffff,   0x9999,     0x0000},
    {0,     0x0000,   0xffff,     0xffff},
    {0,     0x0000,   0x0000,     0x0000},
    {0,     0xffff,   0xffff,     0xffff}
};

/* --- Obliczamy liczbę kolorów --- */
static int liczbaKolorow = sizeof (listakolorow) / sizeof (GdkColor);

/* --- Tworzymy style okien dialogowych --- */
GtkStyle *stylBozeNarodzenie;
GtkStyle *stylWszystkichSwietych;
GtkStyle *stylSwietyPatryk;
GtkStyle *stylDzienNiepodleglosci;

/* --- pionowe pole pakujące na przyciski --- */
GtkWidget *ypole;

/*
 * UtworzKolorowyStyl
 *
 * Tworzy styl na podstawie przekazanych kolorów.
 * Ustawia kolor pierwszoplanowy, kolor tła i kolor
 * tekstu. Zwróćmy uwagę, że wszystkie stany kontrolki
 * będą posiadały te same kolory.
 *
 * kp    - kolor pierwszoplanowy
 * tekst - kolor tekstu
 * kt    - kolor tła
 */
GtkStyle *UtworzKolorowyStyl (GdkColor kp,
                              GdkColor tekst,
                              GdkColor kt)
{
    GtkStyle *styl_dom;
    GtkStyle *styl;
    int i;

    /* --- Pobieramy styl domyślny --- */
```

```
    styl_dom = gtk_widget_get_default_style ();

    /* --- Tworzymy kopię --- */
    styl = gtk_style_copy (styl_dom);

    /* --- Ustawiamy kolory dla każdego stanu --- */
    for (i = 0; i < 5; i++) {

        /* --- Ustawiamy kolory dla stylu --- */
        styl->fg[i] = kp;
        styl->text[i] = tekst;
        styl->bg[i] = kt;
    }

    /* --- Gotowe, oto nowy styl --- */
    return (styl);
}

/*
 * StylLosowy
 *
 * Tworzy losowy styl na podstawie zdefiniowanej wyżej
 * listy kolorów. Nie sprawdza, czy dobór kolorów jest
 * właściwy (np. czy kolor pierwszoplanowy nie jest taki
 * sam jak kolor tła).
 */
GtkStyle *StylLosowy ()
{
    GtkStyle *styl;

    /* --- Tworzymy styl z losowymi kolorami --- */
    styl = UtworzKolorowyStyl (
            listakolorow[random () % liczbaKolorow],
            listakolorow[random () % liczbaKolorow],
            listakolorow[random () % liczbaKolorow]);

    return (styl);
}

/*
 * UtworzOdjazdoweKolory
 *
 * Tworzy style dla przycisków. Każdy styl posiada inny zestaw
```

```
 * kolorów, które będą przypisane do przycisków.
 */
void UtworzOdjazdoweKolory ()
{
   /* --- Czerwony na zielonym --- */
   stylBozeNarodzenie = UtworzKolorowyStyl (czerwony,
                        czerwony, zielony);

   /* --- Pomarańczowy na czarnym --- */
   stylWszystkichSwietych = UtworzKolorowyStyl (pomaranczowy,
                        pomaranczowy, czarny);

   /* --- Zielony na białym --- */
   stylSwietyPatryk = UtworzKolorowyStyl (zielony,
                        zielony, bialy);

   /* --- Czerwony i biały na niebieskim  --- */
   stylDzienNiepodleglosci = UtworzKolorowyStyl (czerwony,
                        bialy, niebieski);
}

/*
 * ZamknijApl
 *
 * Wychodzimy z GTK.
 */
void ZamknijApl (GtkWidget *kontrolka, gpointer gdane)
{
   gtk_main_quit ();
}

/*
 * Kliknięto przycisk
 *
 * procedura obsługi zdarzenia, wywoływana po kliknięciu przycisku
 */
void KliknietoPrzycisk (GtkWidget *kontrolka, gpointer gdane)
{
   GtkStyle *styl;

   /* --- Pobieramy styl z parametru funkcji zwrotnej --- */
   styl = (GtkStyle *) gdane;
```

```
    /* --- Odkładamy na stosie nowy, kolorowy styl - teraz
           staje się stylem domyślnym  --- */
    gtk_widget_push_style (styl);

    /* --- Pokazujemy okno dialogowe w nowym stylu --- */
    WywolajOkno (kontrolka, (gpointer) 2);

    /* --- Usuwamy styl --- */
    gtk_widget_pop_style ();
}

/*
 * UstawionoStyl
 *
 * Wyświetla komunikat po każdej modyfikacji stylu.
 */
void UstawionoStyl (GtkWidget *kontrolka, gpointer dane)
{

    printf ("Ustawiono styl\n");
}

/*
 * UstawStylRekurencyjnie
 *
 * Ustawia styl kontrolki na styl otrzymany w parametrze "dane"
 * i zapewnia, że wszyscy potomkowie kontrolki (jeśli kontrolka
 * jest pojemnikiem) także będą ustawieni na ten styl.
 */
void UstawStylRekurencyjnie (GtkWidget *kontrolka, gpointer dane)
{
    GtkStyle *styl;

    /* --- pobieramy styl --- */
    styl = (GtkStyle *) dane;

    /* --- ustawiamy styl kontrolki --- */
    gtk_widget_set_style (kontrolka, styl);

    /* --- jeśli kontrolka może posiadać potomków --- */
    if (GTK_IS_CONTAINER (kontrolka)) {

        /* --- ustawiamy także styl wszystkich potomków --- */
        gtk_container_foreach (GTK_CONTAINER (kontrolka),
```

```
                        UstawStylRekurencyjnie, styl);
    }
}

/*
 * SzalonyPrzycisk
 *
 * Ustawia kolory przycisku na kolory dobrane losowo
 * ze zdefiniowanej wcześniej tabeli.
 */
void SzalonyPrzycisk (GtkWidget *kontrolka, gpointer dane)
{
    GtkStyle *styl;

    /* --- Wybieramy losowy styl --- */
    styl = StylLosowy ();

    /* --- Ustawiamy styl kontrolki --- */
    UstawStylRekurencyjnie (kontrolka, (gpointer) styl);
}

/*
 * PokolorujPrzyciski
 *
 * Ustawia kolory kontrolek potomnych na losowy styl.
 */
void PokolorujPrzyciski (GtkWidget *kontrolka, gpointer dane)
{
    /* --- Zmieniamy styl wszystkich potomków --- */
    gtk_container_foreach (GTK_CONTAINER (ypole),
                           SzalonyPrzycisk, NULL);
}

/*
 * UtworzPrzyciskKolorowegoOkna
 *
 * Tworzy przycisk w pionowym polu pakującym. Styl będzie
 * przekazywany w funkcji zwrotnej dla przycisku, aby okno
 * dialogowe, otwierane po naciśnięciu przycisku, było tworzone
 * w tym stylu.
 */
void UtworzPrzyciskKolorowegoOkna (GtkWidget *ypole, char *etykieta,
```

```
                        GtkStyle *styl)
{
   GtkWidget *przycisk;

   /* --- Tworzymy przycisk z etykietą --- */
   przycisk = gtk_button_new_with_label (etykieta);

   /* --- Konfigurujemy styl w funkcji zwrotnej "clicked" --- */
   gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                       GTK_SIGNAL_FUNC    (KliknietoPrzycisk),    (gpointer)
styl);

   /* --- Konfigurujemy styl w funkcji zwrotnej "style_set" --- */
   gtk_signal_connect (GTK_OBJECT (przycisk), "style_set",
                       GTK_SIGNAL_FUNC (UstawionoStyl), (gpointer) styl);

   /* --- Umieszczamy przycisk w pionowym polu pakującym --- */
   gtk_box_pack_start (GTK_BOX (ypole), przycisk, FALSE, FALSE, 0);

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

/*
 * UtworzPrzyciskKolorujacy
 *
 * Tworzy przycisk, który zmienia kolory wszystkich przycisków
 */
void UtworzPrzyciskKolorujacy (GtkWidget *ypole, char *etykieta)
{
   GtkWidget *przycisk;

   /* --- Tworzymy przycisk --- */
   przycisk = gtk_button_new_with_label (etykieta);

   /* --- Ustawiamy obsługę sygnału --- */
   gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                       GTK_SIGNAL_FUNC (PokolorujPrzyciski), NULL);

   /* --- Umieszczamy przycisk w polu pakującym --- */
   gtk_box_pack_start (GTK_BOX (ypole), przycisk, FALSE, FALSE, 0);

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

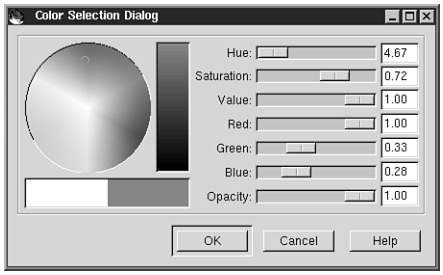
```
}

/*
 * UtworzPrzyciskBlyskajacy
 *
 */
void UtworzPrzyciskBlyskajacy (GtkWidget *ypole, char *etykieta)
{
   GtkStyle *styl_dom;
   GtkStyle *styl;

   styl_dom = gtk_widget_get_default_style ();
   styl = gtk_style_copy (styl_dom);

   styl->fg[GTK_STATE_NORMAL] = fioletowy;
   styl->text[GTK_STATE_NORMAL] = fioletowy;
   styl->bg[GTK_STATE_NORMAL] = turkusowy;

   styl->fg[GTK_STATE_PRELIGHT] = zielony;
   styl->text[GTK_STATE_PRELIGHT] = zielony;
   styl->bg[GTK_STATE_PRELIGHT] = niebieski;

   styl->fg[GTK_STATE_ACTIVE] = pomaranczowy;
   styl->text[GTK_STATE_ACTIVE] = pomaranczowy;
   styl->bg[GTK_STATE_ACTIVE] = zolty;

   gtk_widget_push_style (styl);

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

   /* --- Umieszczamy przycisk w polu pakującym --- */
   gtk_box_pack_start (GTK_BOX (ypole), przycisk, FALSE, FALSE, 0);

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

   /* --- Usuwamy styl, aby nie był stylem domyślnym --- */
   gtk_widget_pop_style ();
}

/*
 * main
 *
```

```
 * Tutaj zaczyna się program
 */
int main (int argc, char *argv[])
{
   GtkWidget *okno;

   /* --- Inicjacja GTK, obsługa parametrów wiersza polecenia --- */
   gtk_init (&argc, &argv);

   /* --- Tworzymy okno gtk - na razie NIE JEST widoczne --- */
   okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

   gtk_signal_connect (GTK_OBJECT (okno), "destroy",
                       GTK_SIGNAL_FUNC (ZamknijApl), NULL);

   /* --- Trochę przestrzeni wokół obiektów w pojemniku --- */
   gtk_container_border_width (GTK_CONTAINER (okno), 15);

   ypole = gtk_vbox_new (FALSE, 0);

   /* --- Przydzielamy kolory --- */
   PrzydzielKolory ();

   /* --- Przydzielamy style --- */
   UtworzOdjazdoweKolory ();

   /* --- Tworzymy przyciski --- */
   UtworzPrzyciskKolorowegoOkna (ypole, "Boże Narodzenie",
                       stylBozeNarodzenie);
   UtworzPrzyciskKolorowegoOkna (ypole, "Dzień Św. Patryka",
                       stylSwietyPatryk);
   UtworzPrzyciskKolorowegoOkna (ypole, "Wszystkich Świętych",
                       stylWszystkichSwietych);
   UtworzPrzyciskKolorowegoOkna (ypole, "Dzień Niepodległości",
                       stylDzienNiepodleglosci);

   UtworzPrzyciskKolorujacy (ypole, "Szalone Przyciski");

   UtworzPrzyciskBlyskajacy (ypole, "Błyskający Przycisk");

   /* --- Teraz uwidaczniamy okno --- */
   gtk_widget_show (ypole);
   gtk_container_add (GTK_CONTAINER (okno), ypole);
   gtk_widget_show (okno);
```

```
    /* --- Pętla zdarzeń w gtk. Czekamy na zakończenie programu --- */
    gtk_main ();

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

Okno dialogowe wyboru koloru

Okno wyboru koloru udostępnia użytkownikom standardową metodę wybierania kolorów, podobnie jak okno wyboru pliku jest standardowym interfejsem, służącym do wybierania plików. Okno wyboru kolorów zawiera kilka kontrolek, które pomagają użytkownikowi w wybieraniu kolorów (patrz rysunek 11.2).



Rysunek 11.2. Kontrolka wyboru koloru.

Okno wyboru koloru tworzymy przy pomocy funkcji gtk_color_selection_dialog_new. Przyciskom OK i Cancel możemy przypisać funkcje zwrotne, które będą obsługiwać zdarzenia związane z przyciskami.

```
/* --- nowe okno dialogowe --- */
okno = gtk_color_selection_dialog_new ("Okno wyboru koloru");

/* --- chcemy wiedzieć, czy użytkownik nacisnął OK --- */
gtk_signal_connect (
   GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (okno)->ok_button),
   "clicked",
   GTK_SIGNAL_FUNC (KliknietoPrzyciskOK),
   daneKoloru);

/* --- usuwamy okno dialogowe po kliknięciu Cancel --- */
gtk_signal_connect (
```

```
    GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (okno)-
>cancel_button),
    "clicked",
    GTK_SIGNAL_FUNC (gtk_widget_destroy),
    GTK_OBJECT(okno));
```

Sygnał color_changed informuje, że zmienił się kolor w oknie dialogowym. Można określić założenia, dotyczące wysyłania komunikatów o zmianie koloru, za pomocą funkcji gtk_color_selection_set_update_policy. Założenia te to GTK_UPDATE_CONTINUOUS, GTK_UPDATE_DISCONTINUOUS i GTK_ UPDATE_DELAYED.

GTK_UPDATE_CONTINUOUS wysyła uaktualnienia w sposób ciągły, w miarę zmieniania się koloru w oknie dialogowym. GTK_UPDATE_DISCONTINUOUS wysyła uaktualnienie po zwolnieniu klawisza myszy. GTK_UPDATE_ DELAYED wysyła uaktualnienie po zwolnieniu klawisza myszy, albo wtedy, kiedy wskaźnik myszy zatrzyma się na pewien czas w kręgu kolorów.

```
/* --- chcemy wiedzieć, kiedy zmieni się kolor --- */
gtk_signal_connect (
    GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (okno)->colorsel),
    "color_changed",
    GTK_SIGNAL_FUNC (ZmienilSieKolor),
    okno);

/* --- chcemy być natychmiast informowani o zmianie koloru --- */
/* --- zostaniemy zalani komunikatami, ale mamy szybki komputer --- */
gtk_color_selection_set_update_policy (
    GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (okno)->colorsel),
    GTK_UPDATE_CONTINUOUS);
```

Kiedy procedura obsługi zdarzenia chce poznać bieżący kolor w oknie dialogowym, musi przeprowadzić kilka czynności. Najpierw musi pobrać pole colorsel ze struktury okna dialogowego. Następnie musi uzyskać tablicę wartości kolorów (zdefiniowanych jako gdouble). Wartości te mają zakres 0...1, więc należy je przekształcić do zakresu 0...0xffff, mnożąc je przez 0xffff. Oprócz zdefiniowanych kolorów, zwrócona wartość może zawierać także przezroczystość (*opacity*) koloru. Można skonfigurować przezroczystość przy pomocy funkcji gtk_color_selection_ set_opacity, która spowoduje utworzenie dodatkowych kontrolek w oknie wyboru koloru, umożliwiających wybór przezroczystości. Przezroczystość jest zdefiniowana jako czwarta wartość w tablicy.

```
GtkColorSelection *wybKoloru;
```

```
gdouble *daneKoloru[4];

/* --- pobieramy okno koloru, aby sprawdzić wybrany kolor --- */
wybKoloru = GTK_COLOR_SELECTION (okno->colorsel);

/* --- pobieramy wartości koloru z kontrolki --- */
gtk_color_slection_get_color (wybKoloru, daneKoloru);

/* --- "kolor" to struktura GdkColor --- */
kolor->red = daneKoloru[0] * 0xffff;
kolor->green = daneKoloru[1] * 0xffff;
kolor->blue = daneKoloru[2] * 0xffff;

/* --- Przezroczystość znajduje się w daneKoloru[3], jeśli ustawiliśmy
       ją w oknie dialogowym --- */

/* --- Kolor został pobrany i jest gotowy do użycia --- */
```

Możemy zilustrować omówione zagadnienia, używając okna wyboru koloru do zmiany koloru kontrolki. Poniższy program umożliwia użytkownikowi zmianę koloru przycisku, poprzez kliknięcie przycisku i wybranie nowego koloru z okna dialogowego. Po kliknięciu przycisku OK styl przycisku zostanie zmodyfikowany tak, aby uwzględniał nowy kolor. Przykład składa się z dwóch plików - kolorowyprz.c i kolordiag.c.

kolorowyprz.c

Plik kolorowyprz.c tworzy główne okno, steruje przyciskami i zmienia kolory.

```
/*
 * Plik: kolorowyprz.c
 *
 */
#include <gtk/gtk.h>
#include <stdlib.h>

GtkWidget *przycisk;
GtkWidget *ypole;

/*
 * UtworzStylTla
 *
 * Tworzy styl na podstawie przekazanych kolorów
 * Ustawia nowy kolor tła. Zwróćmy uwagę, że wszystkie
```

```
 * stany kontrolki będą posiadały ten sam kolor.
 *
 * kt - kolor tła
 */
GtkStyle *UtworzStylTla (GdkColor kt)
{
   GtkStyle *styl_dom;
   GtkStyle *styl;
   int i;

   /* --- Pobieramy styl domyślny --- */
   styl_dom = gtk_widget_get_default_style ();

   /* --- Tworzymy jego kopię --- */
   styl = gtk_style_copy (styl_dom);

   /* --- Ustawiamy kolory dla każdego stanu --- */
   for (i = 0; i < 5; i++) {

      /* --- Ustawiamy kolory dla stanu --- */
      styl->bg[i] = kt;
   }

   /* --- Gotowe, oto nowy styl --- */
   return (styl);
}

/*
 * NowyStyl
 *
 */
GtkStyle *NowyStyl (GdkColor k)
{
   GtkStyle *styl;

   /* --- Tworzymy nowy styl na podstawie koloru --- */
   styl = UtworzStylTla (k);

   /* --- Zwracamy styl --- */
   return (styl);
}

/*
 * ZamknijOknoApl
```

```
 *
 * Zamyka aplikację
 */
gint ZamknijOknoApl (GtkWidget *kontrolka, gpointer gdane)
{
   g_print ("Kończę pracę...\n");
   gtk_main_quit ();
   return (FALSE);
}

/*
 * UstawStylRekurencyjnie
 * Ustawia styl kontrolki i wszystkich jej potomków
 */
void UstawStylRekurencyjnie (GtkWidget *kontrolka, gpointer dane)
{
   GtkStyle *styl;

   /* --- pobieramy styl --- */
   styl = (GtkStyle *) dane;

   /* --- ustawiamy styl kontrolki --- */
   gtk_widget_set_style (kontrolka, styl);

   /* --- jeśli kontrolka może posiadać potomków --- */
   if (GTK_IS_CONTAINER (kontrolka)) {

      /* --- ustawiamy także styl wszystkich potomków --- */
      gtk_container_foreach (GTK_CONTAINER (kontrolka),
                             UstawStylRekurencyjnie, styl);
   }
}

/*
 * KliknietoPrzycisk
 *
 * Procedura obsługi zdarzenia, wywoływana po kliknięciu przycisku.
 * Możemy jej użyć, ponieważ okno wyboru kolorów jest tutaj modalne,
 * i nie oddaje sterowania, dopóki nie zostanie pobrany kolor
 */
void KliknietoPrzycisk (GtkWidget *kontrolka, gpointer gdane)
{
   GtkStyle *styl;
```

```
    GdkColor kolor;

    /* --- Wywołujemy okno dialogowe, aby pobrać kolor --- */
    PobierzKolorZOkna (&kolor);

    /* --- Tworzymy styl na podstawie koloru --- */
    styl = NowyStyl (kolor);

    /* --- Ustawiamy nowy styl kontrolki */
    UstawStylRekurencyjnie (kontrolka, (gpointer) styl);
}

/*
 * UtworzPrzycisk
 *
 * Tworzy przycisk i dodaje go do pionowego pola pakującego.
 * Ustawia także procedurę obsługi zdarzenia na "KliknietoPrzycisk"
 */
void UtworzPrzycisk (GtkWidget *ypole, char *etykieta)
{
    GtkWidget *przycisk;

    /* --- Tworzymy przycisk --- */
    przycisk = gtk_button_new_with_label (etykieta);

    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
            GTK_SIGNAL_FUNC (KliknietoPrzycisk), NULL);

    /* --- Umieszczamy przycisk w polu pakującym  --- */
    gtk_box_pack_start (GTK_BOX (ypole), przycisk, FALSE, FALSE, 0);

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

/*
 * main
 *
 * Tutaj zaczyna się program
 */
int main (int argc, char *argv[])
{
    GtkWidget *okno;
```

```
    /* --- Inicjacja gtk, obsługa parametrów wiersza polecenia --- */
    gtk_init (&argc, &argv);

    /* --- Tworzymy okno --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

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

    /* --- Trochę miejsca wokół obiektów w pojemniku --- */
    gtk_container_border_width (GTK_CONTAINER (okno), 15);

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

    /* --- Tworzymy przycisk --- */
    UtworzPrzycisk (ypole, "Wybierz kolor");

    /* --- Teraz uwidaczniamy okno --- */
    gtk_widget_show (ypole);
    gtk_container_add (GTK_CONTAINER (okno), ypole);
    gtk_widget_show (okno);

    /* --- Nie wracamy aż do zamknięcia aplikacji --- */
    gtk_main ();

    /* --- Kod wyjściowy --- */
    return 0;
}
```

kolordiag.c

Plik kolordiag.c wyświetla okno wyboru koloru i zwraca wybrany kolor.

```
/*
 * Plik: kolordiag.c
 *
 * Wyświetla okno wyboru koloru, w którym użytkownik
 * może określić kolor przycisku.
 */
#include <gtk/gtk.h>

/*
 * Struktura umożliwiająca zrezygnowanie ze zmiennych globalnych
```

```
 */
typedef struct  {

   GtkWidget *dialog;
   GdkColor *kolor;

} typDaneZOknaKolorow;

/*
 * ZamknijDialog
 *
 * Zamyka modalne okno dialogowe przy pomocy funkcji
 * gtk_main_quit, ponieważ uczyniono je modalnym przy pomocy
 * funkcji gtk_main. Zwalnia także pamięć.
 */
void ZamknijDialog (GtkWidget *w, typDaneZOknaKolorow *di)
{
   gtk_main_quit ();
   gtk_grab_remove (w);
   g_free (di);
}

/*
 * KliknietoPrzyciskOK
 *
 * Pobiera kolor zaznaczony w oknie dialogowym i wydobywa
 * jego składowe rgb. Ustawia kolor GdkColor w strukturze
 * daneKolorow na te wartości
 */
void KliknietoPrzyciskOK (GtkWidget *k,
            typDaneZOknaKolorow *daneKolorow)
{
   GtkColorSelectionDialog *wk;
   GtkColorSelection *wyb_kolor;
   gdouble kolor[4];

   /* --- Pobieramy okno dialogowe i kolor --- */
   wk = (GtkColorSelectionDialog *) daneKolorow->dialog;
   wyb_kolor= GTK_COLOR_SELECTION (wk->colorsel);

   /* --- Pobieramy kolor i przypisujemy go do GdkColor --- */
   gtk_color_selection_get_color (wyb_kolor, kolor);
   daneKolorow->kolor->red = kolor[0] * 0xffff;
```

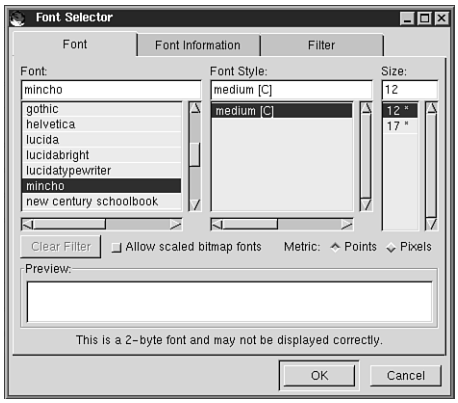
```
    daneKolorow->kolor->green = kolor[1] * 0xffff;
    daneKolorow->kolor->blue = kolor[2] * 0xffff;

    /* --- Usuwamy okno dialogowe --- */
    gtk_widget_destroy (GTK_WIDGET (wk));
}

/*
 * ZmienilSieKolor
 *
 * Funkcja powiadamiana o zmianie koloru w oknie dialogowym
 */
void ZmienilSieKolor(GtkWidget *w, GtkColorSelectionDialog *wk)
{
    GtkColorSelection *wyb_kolor;
    gdouble kolor[4];

    /* --- Pobieramy kolor z zaznaczenia --- */
    wyb_kolor = GTK_COLOR_SELECTION (wk->colorsel);
    gtk_color_selection_get_color (wyb_kolor, kolor);

    g_print ("Zmienił się kolor!\n");
}

/*
 * PobierzKolorZOkna
 *
 * Wyświetla modalne okno dialogowe i umożliwia użytkownikowi
 * wybranie koloru, który zostanie zwrócony.
 */
void PobierzKolorZOkna (GdkColor *kolor)
{
    static GtkWidget *okno = NULL;
    typDaneZOknaKolorow *daneKolorow;

    /* --- Nowe okno dialogowe --- */
    okno = gtk_color_selection_dialog_new ("Okno wyboru koloru");

    /* --- Przydzielamy pamięć na strukturę daneKolorow
           i wypełniamy jej pola --- */
    daneKolorow = g_malloc (sizeof (typDaneZOknaKolorow));
    daneKolorow->dialog = okno;
    daneKolorow->kolor = kolor;
```

```
    gtk_color_selection_set_opacity (GTK_COLOR_SELECTION (
        GTK_COLOR_SELECTION_DIALOG (okno)->colorsel), TRUE);

    gtk_color_selection_set_update_policy (GTK_COLOR_SELECTION (
        GTK_COLOR_SELECTION_DIALOG (okno)->colorsel),
        GTK_UPDATE_CONTINUOUS);

    /* --- Chcemy być informowani o usunięciu okna --- */
    gtk_signal_connect (GTK_OBJECT (okno), "destroy",
                GTK_SIGNAL_FUNC (ZamknijDialog),
                daneKolorow);

    /* --- Chcemy być informowani o zmianie kolorów --- */
    gtk_signal_connect (GTK_OBJECT (
                GTK_COLOR_SELECTION_DIALOG (okno)->colorsel),
                "color_changed",
                GTK_SIGNAL_FUNC (ZmienilSieKolor),
                okno);

    /* --- Chcemy być informowani o naciśnięciu przycisku OK --- */
    gtk_signal_connect (GTK_OBJECT (
                GTK_COLOR_SELECTION_DIALOG (okno)->ok_button),
                "clicked",
                GTK_SIGNAL_FUNC (KliknietoPrzyciskOK),
                daneKolorow);

    /* --- Usuwamy okno dialogowe po kliknięciu Cancel --- */
    gtk_signal_connect_object (GTK_OBJECT (
                GTK_COLOR_SELECTION_DIALOG (okno)->cancel_button),
                "clicked",
                GTK_SIGNAL_FUNC(gtk_widget_destroy),
                GTK_OBJECT (okno));

    /* --- Pokazujemy okno --- */
    gtk_widget_show (okno);

    /* --- Przechwytujemy sterowanie --- */
    gtk_grab_add (okno);

    /* --- To sprawi, że okno będzie modalne, aż do wywołania
        gtk_main_quit --- */
    gtk_main ();
}
```

Czcionki

Czcionki można modyfikować w sposób zbliżony do koloru, ale we wszystkich stanach kontrolki istnieje tylko jedna czcionka. Czcionkę można załadować używając jej nazwy albo przy pomocy okna wyboru czcionki (patrz rysunek 11.3). Jeśli chcemy użyć nowej czcionki nie korzystając z okna wyboru czcionki, musimy znać jej nazwę i użyć funkcji gtk_font_load. Po załadowaniu czcionki można odpowiednio ustawić pole czcionki w strukturze stylu.

```
/* --- ładujemy czcionkę --- */
czcionka = gdk_font_load (szCzcionka);

/* --- przypisujemy czcionkę do stylu --- */
styl->font = czcionka;
```



Rysunek 11.3. Okno wyboru czcionki.

Trzeba więc znaleźć sposób ustalenia nazw czcionek w systemie. Aby wyświetlić wszystkie dostępne w systemie czcionki, można użyć funkcji XListFonts z biblioteki Xlib. Funkcja ta (oprócz innych parametrów) przyjmuje wzorzec, który określa, jakie czcionki zostaną zwrócone w liście nazw czcionek. Aby funkcja zwróciła wszystkie dostępne czcionki, używamy wzorca -*. Gwiazdka * pasuje do dowolnego tekstu, a więc także do dowolnej nazwy czcionki. Nazwy czcionek przypominają te umieszczone poniżej (są to niektóre spośród czcionek, zainstalowanych w systemie autora):

```
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
```

```
-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso8859-1
-daewoo-gothic-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
-daewoo-mincho-medium-r-normal--24-170-100-100-c-240-ksc5601.1987-0
-isas-fangsong ti-medium-r-normal--16-160-72-72-c-160-gb2312.1980-0
-isas-song ti-medium-r-normal--24-240-72-72-c-240-gb2312.1980-0
-jis-fixed-medium-r-normal--24-230-75-75-c-240-jisx0208.1983-0
-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
-adobe-times-medium-i-normal--12-120-75-75-p-63-iso8859-1
```

Poniższy krótki program wyświetla wszystkie czcionki w systemie:

```
/*
 * Plik: PokazCzcionki.c
 *
 * Wyświetla wszystkie obecne w systemie czcionki
 *
 * Trzeba go uruchomić z terminala X
 */

#include <gtk/gtk.h>
#include <gdk/gdkx.h>
#include <X11/Xlib.h>

/*
 * Jeśli użytkownicy mają więcej fontów, niż widać poniżej,
 * to trudno; to i tak sporo za dużo
 */
#define MAKS_CZCIONKI 30000

/*
 * main
 *
 */
int main (int argc, char *argv[])
{
   int nCzcionki;
   char **szaNazwyCzcionek;
   int i;

   /* --- Inicjacja GTK+.  Potrzebna do wywołania GDK_DISPLAY --- */
   gtk_init (&argc, &argv);

   /* --- Pobieramy nazwy czcionek --- */
```

```
    szaNazwyCzcionek = XListFonts (GDK_DISPLAY (), "*",
                                   MAKS_CZCIONKI, &nCzcionki);

    /* --- Sprawdzamy uzyskane liczby --- */
    if (nCzcionki == MAKS_CZCIONKI) {

       /* --- W systemie zainstalowano MNÓSTWO czcionek --- */
       printf ("W systemie jest wiele czcionek. "
             "Nie można wyświetlić wszystkich.");
    }

    /* --- Wyświetlamy czcionki --- */
    for (i = 0; i < nCzcionki; i++) {

       /* --- Pobieramy nazwę --- */
       printf ("%s\n", szaNazwyCzcionek[i]);
    }

    XFreeFontNames (szaNazwyCzcionek);

    return (0);
}
```

Możemy zmodyfikować przykład z oknem kolorów tak, aby użytkownik mógł wybierać czcionkę dla przycisku z okna wyboru czcionki. Przykład ten jest niemal identyczny, jak poprzedni, ale zamiast wybierać kolor dla przycisku, będziemy wybierać dla niego czcionkę. Okno dialogowe zwraca nazwę czcionki, którą można przekazać funkcji gtk_load_font w celu załadowania czcionki. Następnie można uaktualnić styl przycisku. Rysunek 11.3 przedstawia okno wyboru czcionki w działaniu. Zauważmy, że czcionka niekoniecznie musi być czcionką angielską.

```
/*
 * Plik: czcionkaprz.c
 *
 */
#include <gtk/gtk.h>
#include <stdlib.h>

GtkWidget *przycisk;
GtkWidget *ypole;

gchar *PobierzCzcionke ();

/*
```

```
 * NowyStyl
 *
 * Tworzy nowy styl na podstawie przekazanej czcionki
 */
GtkStyle *NowyStyl (GdkFont *c)
{
   GtkStyle *styl;
   GtkStyle *styl_dom;

   /* --- Pobieramy styl domyślny --- */
   styl_dom = gtk_widget_get_default_style ();

   /* ---  Tworzymy jego kopię --- */
   styl = gtk_style_copy (styl_dom);

   styl->font = c;

   /* --- Zwracamy styl --- */
   return (styl);
}

/*
 * ZamknijOknoApl
 *
 * Zamyka aplikację
 */
gint ZamknijOknoApl (GtkWidget *kontrolka, gpointer gdane)
{
   g_print ("Kończę pracę...\n");
   gtk_main_quit ();
   return (FALSE);
}

/*
 * UstawStylRekurencyjnie
 * Ustawia styl kontrolki i wszystkich jej potomków
 */
void UstawStylRekurencyjnie (GtkWidget *kontrolka, gpointer dane)
{
   GtkStyle *styl;

   /* --- pobieramy styl --- */
   styl = (GtkStyle *) dane;
```

```
    /* --- ustawiamy styl kontrolki --- */
    gtk_widget_set_style (kontrolka, styl);

    /* --- jeśli kontrolka może posiadać potomków --- */
    if (GTK_IS_CONTAINER (kontrolka)) {

        /* --- ustawiamy także styl wszystkich potomków --- */
        gtk_container_foreach (GTK_CONTAINER (kontrolka),
                               UstawStylRekurencyjnie, styl);
    }
}

/*
 * KliknietoPrzycisk
 *
 * Procedura obsługi zdarzenia, wywoływana po kliknięciu przycisku.
 * Możemy jej użyć, ponieważ okno wyboru czcionki jest tutaj modalne,
 * i nie oddaje sterowania, dopóki nie zostanie pobrany czcionka
 */
void KliknietoPrzycisk (GtkWidget *kontrolka, gpointer gdane)
{
    GtkStyle *styl;
    char *szCzcionka;
    GdkFont *czcionka;

    /* --- Wywołujemy okno dialogowe, aby pobrać czcionkę --- */
    szCzcionka = PobierzCzcionke ();

    printf ("PobierzCzcionke=%s\n", szCzcionka);
    czcionka = gdk_font_load (szCzcionka);
    g_free (szCzcionka);

    /* --- Tworzymy styl na podstawie koloru --- */
    styl = NowyStyl (czcionka);

    /* --- Ustawiamy nowy styl kontrolki */
    UstawStylRekurencyjnie (kontrolka, (gpointer) styl);
}

/*
 * UtworzPrzycisk
 *
 * Tworzy przycisk i dodaje go do pionowego pola pakującego.
```

```
 * Ustawia także procedurę obsługi zdarzenia na "KliknietoPrzycisk"
 */
void UtworzPrzycisk (GtkWidget *ypole, char *etykieta)
{
   GtkWidget *przycisk;

   /* --- Tworzymy przycisk --- */
   przycisk = gtk_button_new_with_label (etykieta);

   gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
            GTK_SIGNAL_FUNC (KliknietoPrzycisk), NULL);

   /* --- Umieszczamy przycisk w polu pakującym  --- */
   gtk_box_pack_start (GTK_BOX (ypole), przycisk, FALSE, FALSE, 0);

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

/*
 * main
 *
 * Tutaj zaczyna się program
 */
int main (int argc, char *argv[])
{
   GtkWidget *okno;

   /* --- Inicjacja gtk, obsługa parametrów wiersza polecenia --- */
   gtk_init (&argc, &argv);

   /* --- Tworzymy okno --- */
   okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

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

   /* --- Trochę miejsca wokół obiektów w pojemniku --- */
   gtk_container_border_width (GTK_CONTAINER (okno), 15);

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

```
    /* --- Tworzymy przyciski --- */
    UtworzPrzycisk (ypole, "Wybierz czcionkę");
    UtworzPrzycisk (ypole, "Zażółć gęślą jaźń");

    /* --- Teraz uwidaczniamy okno --- */
    gtk_widget_show (ypole);
    gtk_container_add (GTK_CONTAINER (okno), ypole);
    gtk_widget_show (okno);

    /* --- Nie wracamy aż do zamknięcia aplikacji --- */
    gtk_main ();

    /* --- Kod wyjściowy --- */
    return 0;
}
```

Poniżej znajduje się kod, tworzący okno wyboru czcionki. Po wywołaniu funkcji PobierzCzcionke zwraca nazwę czcionki w łańcuchu gchar *, który powinien zostać zwolniony, kiedy nie będzie już potrzebny.

```
/*
 * Plik: wybczcionki.c
 *
 * Interfejs okna wyboru czcionki. Tworzy modalne okno dialogowe,
 * w którym można wybrać czcionkę. Funkcja PobierzCzcionke zwraca
 * nazwę czcionki.
 */

#include <gtk/gtk.h>

gchar *szNazwaCzcionki = NULL;

/*
 * KliknietoOK
 *
 * Kliknięto przycisk OK
 */
void KliknietoOK (GtkWidget *kontrolka, GtkWidget *wybcz)
{

    /* --- Rzutujemy na właściwy typ --- */
    GtkFontSelectionDialog *owc = GTK_FONT_SELECTION_DIALOG
(wybcz);

    /* --- Pobieramy nazwę czcionki --- */
```

```
    szNazwaCzcionki = gtk_font_selection_dialog_get_font_name (owc);

    /* --- Wyświetlamy nazwę czcionki --- */
    printf ("Kliknięto OK - %s\n", szNazwaCzcionki);

    /* --- Usuwamy okno wyboru czcionki --- */
    gtk_widget_destroy (wybcz);
}

/*
 * Zamknij
 *
 * Kończy program
 */
void Zamknij (GtkWidget *kontrolka, gpointer dane)
{
    gtk_main_quit ();
}

/*
 * PobierzCzcionke
 *
 * Wyświetla okno wyboru czcionki i umożliwia użytkownikowi
 * wybranie czcionki. Zwraca łańcuch gchar * z nazwą czcionki.
 */
gchar *PobierzCzcionke ()
{
    GtkWidget *kontrolka;
    GtkFontSelectionDialog *wybcz;
    szNazwaCzcionki = NULL;

    /* --- Tworzymy okno wyboru czcionki --- */
    kontrolka = gtk_font_selection_dialog_new("Okno wyboru czcionki");

    /* --- Rzutujemy na właściwy typ --- */
    wybcz = GTK_FONT_SELECTION_DIALOG (kontrolka);

    /* --- Funkcja zwrotna dla przycisku OK --- */
    gtk_signal_connect (GTK_OBJECT (wybcz->ok_button), "clicked",
                        GTK_SIGNAL_FUNC (KliknietoOK), wybcz);

    /* --- Funkcja zwrotna dla przycisku Cancel --- */
    gtk_signal_connect_object (GTK_OBJECT (wybcz->cancel_button),
```

```
                "clicked",
                  GTK_SIGNAL_FUNC (gtk_widget_destroy),
                  GTK_OBJECT (wybcz));

    /* --- Sygna³ destroy --- */
    gtk_signal_connect (GTK_OBJECT (wybcz), "destroy",
                GTK_SIGNAL_FUNC (Zamknij), wybcz);

    /* --- Pokazujemy okno dialogowe --- */
    gtk_widget_show (kontrolka);

    /* --- Modalne - czekamy, aż zostanie usunięte --- */
    gtk_main ();

    /* --- Zwracamy nazwę czcionki --- */
    return (szNazwaCzcionki);
}
```

Wskaźniki myszy

Wskaźniki można wykorzystać do przekazania użytkownikom różnych informacji. Wskaźnik zmienia kształt na przykład wtedy, kiedy przesuniemy go nad pole wejściowe (przypomina dużą literę I) albo podczas zmiany rozmiaru okna (wskaźnik w kształcie strzałek). Czasem tego rodzaju wskazówki mają duże znaczenie. GTK+ zawiera spory zbiór wbudowanych wskaźników; pełną listę można zobaczyć w podkatalogu GDK katalogu instalacyjnego GTK+. Plik gdkcursors.h zawiera definicje wbudowanych wskaźników, które możemy wykorzystać we własnych programach, na przykład GDK_ARROW albo GDK_CLOCK. Wbudowany wskaźnik tworzymy przy pomocy funkcji gdk_cursor_new, przekazując jej identyfikator wskaźnika. Wskaźnik przypisuje się do okna, a nie do kontrolki, jednak łatwo jest ustalić okno, z którym związana jest kontrolka, sprawdzając pole window struktury kontrolki.

```
/* --- tworzymy nowy wskaźnik --- */
wskaznik = gdk_cursor_new (GDK_CROSS);

/* --- przypisujemy wskaźnik do przycisku --- */
gdk_window_set_cursor (przycisk->window, wskaznik);
```

Dosyć proste, ale GTK+ dysponuje ograniczoną liczbą wbudowanych wskaźników. A gdybyśmy zechcieli stworzyć własny wskaźnik? Można to zrobić przy pomocy danych xpm, tak jak w przypadku ikon dla przy-

cisków. Trik polega na tym, że rysunek musi zawierać trzy kolory: jeden dla pierwszego planu, jeden dla tła, a jeden dla koloru przezroczystego. Dane xpm należy przekształcić na dwie bitmapy GdkBitmap-jedną dla pierwszego planu, a drugą dla tła. Po przeprowadzeniu konwersji utworzenie wskaźnika nie przedstawia żadnych problemów. Rozważmy na przykład następujący rysunek xpm:

```
static char *wskaznik_dlon[] = {
"32 32 3 1",
"    c None",
".   c #000000",
"+  c #FFFFFF",
"                              ",
"                              ",
"          ...                 ",
"         .+++.   ...          ",
"         .+++.  .+++.   ...   ",
"   ...   .+++.  .+++.  .+++.  ",
"  .+++.   .+++. .+++.  .+++.   ",
"   .+++.   .+++. .+++. .+++.   ",
"    .+++.  .+++. .+++. .+++.   ",
"     .+++. .+++..+++. .+++.    ",
"      .+++..+++..+++. .+++.    ",
"       .+++.+++..+++..+++.     ",
"   ...  .+++++++++++++++.     ",
"  .+++. .++++++++++++++.      ",
"  .++++..++++++++++++++.      ",
"   ..++++++++++++++++++.      ",
"     .++++++++++++++++++.     ",
"      .++++++++++++++++.      ",
"       ..+++++++++++++.       ",
"         .++++++++++++.       ",
"          .++++++++++.        ",
"           ..........         ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
"                              ",
```

```
"                       ",
"                       "};
```

Jest to trzykolorowy rysunek dłoni, o wysokości i szerokości 32 pikseli. Kolory zdefiniowane w rysunku są ignorowane. Na podstawie tych danych możemy uzyskać rysunek pierwszoplanowy i maskę, używając funkcji zapożyczonej z programu gnumeric (czy oprogramowanie o otwartym źródle nie jest wspaniałe?). Funkcja ta przyjmuje dane xpm i zwraca dwie bitmapy: jedną dla rysunku pierwszoplanowego, a drugą dla rysunku maski.

```
void utworz_bitmape_i_maske_z_xpm (GdkBitmap **bitmapa,
                        GdkBitmap **maska, gchar **xpm)
{
    int wysokosc, szerokosc, kolory;
    char bufor_piksmapy [(32 * 32)/8];
    char bufor_maski [(32 * 32)/8];
    int x, y, piks;
    int kolor_przezrocz, kolor_czarny;

    sscanf (xpm [0], "%d %d %d %d", &wysokosc, &szerokosc,
          &kolory, &piks);

    g_assert (wysokosc == 32);
    g_assert (szerokosc  == 32);
    g_assert (kolory == 3);

    kolor_przezrocz = ' ';
    kolor_czarny = '.';

    for (y = 0; y < 32; y++){
        for (x = 0; x < 32;){
            char wartosc = 0, wartmaski = 0;

            for (piks = 0; piks < 8; piks++, x++){
                if (xpm [4+y][x] != kolor_przezrocz){
                        wartmaski |= 1 << piks;

                        if (xpm [4+y][x] != kolor_czarny){
                                wartosc |= 1 << piks;
                        }
                }
            }
            bufor_piksmapy [(y * 4 + x/8)-1] = wartosc;
```

```
                bufor_maski [(y * 4 + x/8)-1] = wartmaski;
            }
        }
        *bitmapa = gdk_bitmap_create_from_data (NULL,
                                    bufor_piksmapy, 32, 32);
        *maska   = gdk_bitmap_create_from_data (NULL,
                                    bufor_maski, 32, 32);
}
```

Możemy teraz wykorzystać tę funkcję do utworzenia wskaźnika. Oczywiście, wskaźnik należy utworzyć przy pomocy funkcji gdk_cursor_new_from_pixmap, a nie gdk_cursor_new. Funkcja ta przyjmuje dane rysunku i maski, kolory pierwszego planu i tła oraz aktywny punkt (hot point) wskaźnika. *Aktywnym punktem* nazywamy ten punkt wskaźnika, który określa, w którym dokładnie miejscu nastąpiło kliknięcie. W przypadku standardowego wskaźnika - strzałki aktywny punkt leży zazwyczaj w lewym górnym rogu (0, 0), natomiast wskaźnik w kształcie dłoni może mieć aktywny punkt pod palcem wskazującym (zależy to od ułożenia dłoni).

```
GdkBitmap *bitmapa;
GdkBitmap *maska;
GdkColor bialy = {0, 0xffff, 0xffff, 0xffff};
GdkColor czarny = {0, 0x0000, 0x0000, 0x0000};

/* --- przekształcamy dane rysunku (powyżej) na bitmapy --- */
utworz_bitmape_i_maske_z_xpm (&bitmapa, &maska, wskaznik_dlon);

/* --- tworzymy wskaźnik na podstawie rysunków --- */
wskaznik = gdk_cursor_new_from_pixmap (bitmapa, maska, &bialy,
                        &czarny, 8, 8);

/* --- przypisujemy wskaźnik-dłoń do okna --- */
gdk_window_set_cursor (przycisk->okno, wskaznik)
```

W przykładzie tym używamy koloru białego i czarnego, ale kolory wskaźnika mogą być dowolne - jednak nie więcej, niż dwa. Moglibyśmy stworzyć pomarańczową rękę z karmazynową obwódką, przekazując inne kolory do funkcji gdk_cursor_new_from_pixmap. Chociaż można nadać wskaźnikowi myszy dowolnie niegustowne kolory, lepiej ograniczyć się do czarnego i białego.

Poniżej zamieszczamy pełny przykładowy program, którego okno zawiera trzy przyciski. Pierwszy z nich posiada własny wskaźnik w kształcie dłoni, drugi wbudowany wskaźnik w kształcie krzyża, a trzeci - wskaź-

nik w kształcie karabinowej muszki. Trzeci wskaźnik ilustruje użycie innych kolorów, niż czarnego i białego.

```
/*
 * wskazniki.c
 * Autor: Eric Harlow
 *
 * Ilustracja sposobu zmiany wskaźnika myszy
 */

#include <gtk/gtk.h>

static char * wskaznik_dlon[] = {
"32 32 3 1",
"    c None",
".   c #000000",
"+   c #FFFFFF",
"                            ",
"                            ",
"           ...              ",
"          .+++.   ...       ",
"          .+++.  .+++.   ...  ",
"   ...    .+++.  .+++.  .+++.  ",
"  .+++.    .+++. .+++.  .+++.   ",
"   .+++.   .+++. .+++. .+++.    ",
"    .+++.  .+++. .+++. .+++.    ",
"     .+++. .+++..+++. .+++.     ",
"      .+++..+++..+++. .+++.     ",
"       .+++.+++..+++..+++.      ",
"   ...  .++++++++++++++++.      ",
"  .+++. .++++++++++++++++.      ",
"  .++++..++++++++++++++.        ",
"   ..++++++++++++++++++.        ",
"     .++++++++++++++++++.       ",
"      .++++++++++++++++.        ",
"       ..++++++++++++++.        ",
"         .++++++++++++.         ",
"          .++++++++++.          ",
"           ..........           ",
"                            ",
"                            ",
"                            ",
```

```
"                        ",
"                        ",
"                        ",
"                        ",
"                        ",
"                        ",
"                        ",
"                        "};

static char * wskaznik_muszka[] = {
"32 32 3 1",
"    c None",
".   c #000000",
"+  c #FF0000",
"                        ",
"........................ ",
".         +         . ",
".         +         . ",
".        +++++        . ",
".         +         . ",
".      ++++++++++      . ",
".         +         . ",
".        +++++        . ",
".         +         . ",
".      ++++++++++      . ",
".   +  +   +   +  +   . ",
".   +  + +++++ +  +   . ",
". + + + + +  + + + + + . ",
". + + + + +  + + + + + . ",
".+++++++++++++++++++++++++. ",
". + + + + +  + + + + + . ",
". + + + + +  + + + + + . ",
".   +  + +++++ +  +   . ",
".   +  +   +   +  +   . ",
".      ++++++++++      . ",
".         +         . ",
".        +++++        . ",
".         +         . ",
".      ++++++++++      . ",
".         +         . ",
".        +++++        . ",
".         +         . ",
```

```
".          +          . ",
" ..........................  ",
"                           ",
"                           ",
"                           ",
"                           "};

GdkColor bialy = {0, 0xffff, 0xffff, 0xffff};
GdkColor czarny = {0, 0x0000, 0x0000, 0x0000};
GdkColor czerwony = {0, 0xffff, 0x0000, 0x0000};
GdkColor niebieski = {0, 0x0000, 0x0000, 0xffff};

void utworz_bitmape_i_maske_z_xpm (GdkBitmap **bitmapa,
                        GdkBitmap **maska, gchar **xpm)
{
    int wysokosc, szerokosc, kolory;
    char bufor_piksmapy [(32 * 32)/8];
    char bufor_maski [(32 * 32)/8];
    int x, y, piks;
    int kolor_przezrocz, kolor_czarny;

    sscanf (xpm [0], "%d %d %d %d", &wysokosc, &szerokosc,
          &kolory, &piks);

    g_assert (wysokosc == 32);
    g_assert (szerokosc  == 32);
    g_assert (kolory == 3);

    kolor_przezrocz = ' ';
    kolor_czarny = '.';

    for (y = 0; y < 32; y++){
        for (x = 0; x < 32;){
            char wartosc = 0, wartmaski = 0;

            for (piks = 0; piks < 8; piks++, x++){
                if (xpm [4+y][x] != kolor_przezrocz){
                        wartmaski |= 1 << piks;

                        if (xpm [4+y][x] != kolor_czarny){
                                wartosc |= 1 << piks;
                        }
                }
            }
```

```
                bufor_piksmapy [(y * 4 + x/8)-1] = wartosc;
                bufor_maski [(y * 4 + x/8)-1] = wartmaski;
            }
        }
        *bitmapa = gdk_bitmap_create_from_data (NULL,
                                    bufor_piksmapy, 32, 32);
        *maska   = gdk_bitmap_create_from_data (NULL,
                                    bufor_maski, 32, 32);
}

/*
 * ZamknijOknoApl
 *
 * Kończy pracę GTK+ kiedy użytkownik zamknie okno.
 */
gint ZamknijOknoApl (GtkWidget *kontrolka, gpointer *dane)
{
    gtk_main_quit ();
    return (FALSE);
}

/*
 * UtworzTabele
 *
 * Tworzy okno najwyższego poziomu z różnymi opcjami. Ponieważ
 * będziemy to robić wiele razy, najlepiej jest umieścić kod
 * w oddzielnej funkcji.
 */
void UtworzTabele (char *szTytul, gint xopcje, gint yopcje)
{
    GtkWidget *okno;
    GtkWidget *przyciskDlon;
    GtkWidget *przyciskMuszka;
    GtkWidget *przyciskKrzyz;
    GtkWidget *tabela;
    GdkBitmap *bitmapa;
    GdkBitmap *maska;
    GdkCursor *wskaznik;

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

```
/* --- Nadajemy mu tytuł --- */
gtk_window_set_title (GTK_WINDOW (okno), szTytul);

gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
          GTK_SIGNAL_FUNC (ZamknijOknoApl), NULL);

/* --- Ustawiamy obramowanie okna --- */
gtk_container_border_width (GTK_CONTAINER (okno), 10);

/* --- Tworzymy tabelę pakującą 4 x 4 --- */
tabela = gtk_table_new (2, 2, TRUE);

przyciskDlon = gtk_button_new_with_label ("D³oñ");
GTK_TABLE_attach (GTK_TABLE (tabela), przyciskDlon, 0, 1, 0, 1,
               xopcje, yopcje, 0, 0);
gtk_widget_show (przyciskDlon);

przyciskKrzyz = gtk_button_new_with_label ("Krzy¿");
GTK_TABLE_attach (GTK_TABLE (tabela), przyciskKrzyz, 0, 1, 1, 2,
               xopcje, yopcje, 0, 0);
gtk_widget_show (przyciskKrzyz);

przyciskMuszka = gtk_button_new_with_label ("Muszka");
GTK_TABLE_attach (GTK_TABLE (tabela), przyciskMuszka, 1, 2, 0, 1,
               xopcje, yopcje, 0, 0);
gtk_widget_show (przyciskMuszka);

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

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

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

utworz_bitmape_i_maske_z_xpm (&bitmapa, &maska, wskaznik_dlon);
wskaznik = gdk_cursor_new_from_pixmap (bitmapa, maska, &bialy,
                                       &czarny, 16, 8);
gdk_window_set_cursor (przyciskDlon->window, wskaznik);

utworz_bitmape_i_maske_z_xpm (&bitmapa, &maska, wskaznik_muszka);
wskaznik = gdk_cursor_new_from_pixmap (bitmapa, maska, &czerwony,
                                       &niebieski, 16, 16);
```

```
    gdk_window_set_cursor (przyciskMuszka->window, wskaznik);

    wskaznik = gdk_cursor_new (GDK_CROSS);
    gdk_window_set_cursor (przyciskKrzyz->window, wskaznik);
}

/*
 * --- main
 *
 * Tutaj zaczyna się program
 *
 * Zamiast powielać kod, wywołujemy funkcję UtworzTabele.
 * Wykona ona całą pracę - określamy tylko, jak powinno
 * wyglądać okno.
 */
int main (int argc, char *argv[])
{

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

    /* --- Nie ustawiamy żadnych znaczników --- */
    UtworzTabele ("Brak znaczników", 0, 0);

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

    exit (0);
}
```

Referencje

Każde działanie w GTK+ wiąże się z przydzielaniem elementów. Style są tworzone, albo kopiowane z już istniejących. Tworzone są kontrolki, a czcionki są ładowane i przypisywane do stylów. Powtarzanie tych procesów doprowadziłoby w końcu do wyczerpania się pamięci komputera - gdyby nie istniał sposób łatwego zarządzania jej użyciem. GTK+ zarządza używaną przez siebie pamięcią przy pomocy referencji (odwołań) do obiektów.

Referencja do obiektu zapobiega jego usunięciu. Za każdym razem, kiedy jakiś obiekt (na przykład czcionka) jest przypisywany do innego obiektu przez procedury GTK+, jego licznik referencji jest zwiększany o 1. Kiedy

jakiś obiekt zostanie usunięty, GTK+ sprawdza wszystkie obiekty, do których się odwoływał i zmniejsza ich liczniki referencji. Jeśli licznik referencji jest równy zero, GTK+ zakłada, że obiekt jest już niepotrzebny i zwalnia go.

Większość tych działań odbywa się w sposób przezroczysty i zazwyczaj nie stanowi problemu w przypadku niewielkich aplikacji. Jeśli jednak będziemy nieostrożni, w większych programach możemy mieć kłopoty z „wyciekaniem" pamięci. Kiedy tworzone są kontrolki, ich licznik referencji początkowo wynosi zero. Jeśli zostaną w ten czy inny sposób przypisane do pojemnika, licznik referencji jest zwiększany o 1, aby zaznaczyć, że pojemnik odwołuje się do kontrolki. Kiedy pojemnik jest usuwany, zmniejsza on licznik referencji wszystkich kontrolek, do których się odwołuje - czyli wszystkich obiektów w pojemniku. Jeśli dla którejkolwiek kontrolki licznik referencji spadnie do zera, kontrolka zostanie usunięta. Ma to sens, ponieważ w tym momencie kontrolka nie powinna już być używana. Można zapobiec usunięciu kontrolki, zwiększając jej licznik referencji; w takim przypadku po usunięciu pojemnika kontrolka nadal będzie istnieć. Oczywiście, nie będzie z niej żadnego pożytku, o ile nie zachowamy wskaźnika do kontrolki, aby użyć go w innym miejscu programu.

Obiekty nie będące kontrolkami są traktowane nieco inaczej. Po utworzeniu ich licznik referencji jest automatycznie ustawiany na jeden. Obiekty takie zazwyczaj reprezentują style, piksmapy lub inne wielokrotnie używane elementy, które chcemy przydzielić raz, a wykorzystać w programie kilka razy. Nie chcemy, aby aplikacja usunęła stworzony przez nas styl tylko dlatego, że usunęliśmy ostatnią kontrolkę, która go używała. Jednak często musimy usunąć referencję do stworzonych przez nas elementów jednorazowego użytku, w następujący sposób:

```
/* --- tworzymy piksmapę na podstawie danych --- */
piksmapa = gdk_pixmap_create_from_xpm_d (
                    glowne_okno->window,
                    &maska, NULL, (gchar **) dane_xpm);

/* --- tworzymy kontrolkę piksmapy --- */
kontrolka = gtk_pixmap_new (piksmapa, NULL);

/* --- nie potrzebujemy już piksmapy --- */
gdk_pixmap_unref (piksmapa);
```

Usunięcie referencji do piksmapy zazwyczaj spowodowałoby usunięcie piksmapy, ale tutaj, w rezultacie działania funkcji gtk_pixmap_new, do piksmapy odwołuje się kontrolka. Licznik referencji piksmapy zostanie

zmniejszony, kiedy usuniemy kontrolkę, a jeśli spadnie do zera, piksmapa również zostanie usunięta.

Podsumowanie

Style składają się z kolorów i czcionek i określają wygląd kontrolek. Style można modyfikować indywidualnie, albo można ustalić styl domyślny dla nowo tworzonych kontrolek. GTK+ posiada wbudowany zbiór wskaźników myszy, dzięki którym można modyfikować wygląd wskaźnika myszy w obrębie kontrolki. Można używać wbudowanych wskaźników, ale można także łatwo stworzyć własne. GTK+ używa referencji, które pomagają w zarządzaniu obiektami i zwalnia obiekty, których licznik referencji spadnie do zera.