Rozdział 7

Aplikacja kalkulatora

W rozdziale tym wykorzystamy zdobytą wiedzę, aby stworzyć prosty kalkulator. Aplikacja tego typu stanowi dobry przykład, ponieważ składa się z grupy przycisków i pola wyświetlającego rezultaty. Przyciski ułożone są w tabeli pakującej, pod polem rezultatów. Aplikacja pozwala posługiwać się kalkulatorem zarówno przy pomocy myszy, jak i klawiatury i przeprowadza podstawowe działania arytmetyczne.

Program kalkulatora

Istotnym aspektem pracy kalkulatora jest wizualne potwierdzanie operacji, przeprowadzanych przez użytkowników. Jeśli użytkownik kliknie przycisk kalkulatora, powinien mieć pewność, że przycisk został wciśnięty - to znaczy, przycisk powinien opuścić się i podnieść z powrotem. Oczywiście, pole rezultatów powinno zostać uaktualnione odpowiednio do klikniętego przycisku. Jednakże niektórzy użytkownicy wolą posługiwać się klawiaturą, niż myszą - oni również powinni otrzymać potwierdzenie swoich czynności. Nasza aplikacja będzie przesuwać ognisko na ekranowy przycisk, odpowiadający klawiszowi na klawiaturze. Dzięki temu użytkownicy wprowadzający dane w ten sposób również zobaczą podświetlenie przycisku i uaktualnione pole rezultatów.

Struktury danych

Każdy przycisk w kalkulatorze posiada zbiór związanych z nim właściwości: wskaźnik do kontrolki, etykietę oraz pozycję w rzędzie i kolumnie, która określa, gdzie należy go umieścić wewnątrz tabeli pakującej.

Struktura danych dla każdego przycisku wygląda następująco:

```
/*
 * --- struktura przechowująca dane o przyciskach
```

```
 */
typedef struct {
  char    *szEtykieta;  /*---etykieta wyświetlana na przycisku---*/
  int     wiersz;    /*---rząd, w którym należy umieścić przycisk---*/
  int     kol;      /*---kolumna, w którym należy umieścić przycisk---*/
  GtkWidget *kontrolka; /*---wskaźnik do przycisku---*/
} typPrzyciskKalkulatora;
```

Użyjemy tej struktury do zdefiniowania tablicy przycisków. Jest to tablica elementów typu typPrzyciskKalkulatora. Korzyścią, jaką odnosimy definiując przyciski w ten sposób, jest możliwość łatwego dodawania lub usuwania przycisków bez znaczących zmian w kodzie. Możemy także dowolnie przestawiać przyciski w oknie, nie zmieniając reszty programu. Przyciski kalkulatora są zdefiniowane następująco:

```
typPrzyciskKalkulatora listaPrzyciskow [] = {
   {"C",        1, 0,    NULL},      /* --- Czyszczenie --- */
   {"CE",       1, 1,    NULL},      /* --- Czyszczenie --- */
   {"/",        1, 3,    NULL},      /* --- Dzielenie --- */

   {"7",        2, 0,    NULL},      /* --- Cyfra --- */
   {"8",        2, 1,    NULL},      /* --- Cyfra --- */
   {"9",        2, 2,    NULL},      /* --- Cyfra --- */
   {"*",        2, 3,    NULL},      /* --- Dzielenie --- */
   {"%",        2, 4,    NULL},      /* --- Procent --- */

   {"4",        3, 0,    NULL},      /* --- Cyfra --- */
   {"5",        3, 1,    NULL},      /* --- Cyfra --- */
   {"6",        3, 2,    NULL},      /* --- Cyfra --- */
   {"-",        3, 3,    NULL},      /* --- Odejmowanie --- */
   {"1/x",      3, 4,    NULL},      /* --- 1/x --- */

   {"1",        4, 0,    NULL},      /* --- Cyfra --- */
   {"2",        4, 1,    NULL},      /* --- Cyfra --- */
   {"3",        4, 2,    NULL},      /* --- Cyfra --- */
   {"+",        4, 3,    NULL},      /* --- Dodawanie --- */
   {"pierw",    4, 4,    NULL},      /* --- Pierwiastek --- */

   {"+/-",      5, 0,    NULL},      /* --- Zanegowanie --- */
   {"0",        5, 1,    NULL},      /* --- zero --- */
   {".",        5, 2,    NULL},      /* --- Kropka dziesiętna --- */
   {"=",        5, 3,    NULL},      /* --- Równa się/suma --- */
   {"x^2",      5, 4,    NULL},      /* --- Kwadrat --- */
```

```
};
```

Poniższy wzór oblicza liczbę przycisków w tablicy:

```
int nPrzyciski = sizeof (listaPrzyciskow) /
                 sizeof (typPrzyciskKalkulatora)
```

Wzór ten automatycznie oblicza liczbę przycisków w tablicy w czasie kompilacji, abyśmy nie musieli ustawiać jej ręcznie po dodaniu albo usunięciu przycisku. Jest to z pewnością wygodniejsza metoda, niż użycie stałej opisującej rozmiar tablicy, którą należałoby uaktualniać za każdym razem, kiedy dodalibyśmy albo usunęli przycisk kalkulatora.

Oprócz tablicy z informacjami o przyciskach (listaPrzyciskow) aplikacja potrzebuje także innych zmiennych globalnych, na przykład wskaźnika do etykiety, używanej do wyświetlania rezultatów. Wyświetlacz LCD kalkulatora jest reprezentowany przez etykietę, ponieważ kontrolowanie wpisywanych danych jest o wiele łatwiejsze, jeśli użytkownik nie może wpisywać ich bezpośrednio do kontrolki. Wszystkie naciśnięcia klawiszy są przechwytywane przez okno aplikacji. Okno aplikacji uaktualnia etykietę po odfiltrowaniu znaków, które nie powinny się w niej znaleźć.

Główny program

Po wyjaśnieniu założeń dotyczących kalkulatora oraz opisie struktur danych możemy zacząć pisanie kodu. Funkcja main przeprowadza czynności wstępne, konieczne do uruchomienia kalkulatora. Należy utworzyć i rozmieścić okna oraz przypisać odpowiednie funkcje zwrotne, a następnie wywołać funkcję gtk_main, aby rozpocząć przetwarzanie sygnałów. Zaczynamy od inicjacji GTK+ i utworzenia okna aplikacji.

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

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

   /* --- tworzymy okno kalkulatora --- */
   okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

Powinniśmy nadać aplikacji opisowy tytuł i ustawić minimalny rozmiar okna, ponieważ domyślne okno jest nieco za małe, aby użytkownik mógł wygodnie klikać przyciski.

```
/* --- Nadajemy oknu tytuł --- */
gtk_window_set_title (GTK_WINDOW (okno), "Kalkulator");

/* --- Ustawiamy minimalny rozmiar okna --- */
gtk_widget_set_usize (okno, 200, 200);
```

Jak zwykle, nie należy zapominać o przechwyceniu sygnału, informującego o zamykaniu okna aplikacji. Okno aplikacji oczekuje także na sygnał key_press_event, aby użytkownik mógł posługiwać się klawiaturą, a nie tylko myszą. Wciśnięcia klawiszy muszą zostać odwzorowane na odpowiednie przyciski, przy użyciu zdefiniowanej wcześniej tablicy, i obsłużone tak, jakby użytkownik kliknął dany przycisk myszą.

```
/* --- musimy wiedzieć, że naciśnięto klawisz  --- */
gtk_signal_connect (GTK_OBJECT (okno), "key_press_event",
                    GTK_SIGNAL_FUNC (wcisnieto_klawisz), NULL);

/* --- Zawsze należy podłączyć sygnał delete_event
 *     do głównego okna. --- */
gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                    GTK_SIGNAL_FUNC (ZamknijOknoApl), NULL);
```

Tabela pakująca, w której umieścimy przyciski, ma rozmiary 5 x 5, przy czym pierwszy rząd służy do przechowywania wyświetlacza. Funkcja UtworzPrzyciskiKalkulatora odczytuje tablicę przycisków i rozmieszcza je we właściwych punktach tabeli pakującej. Dodaje także wspólną procedurę obsługi zdarzenia dla wszystkich przycisków, która będzie wywoływana w razie kliknięcia przycisku albo naciśnięcia klawisza.

```
/* --- Tworzymy tabelę 5x5 dla elementów kalkulatora  --- */
tabela = gtk_table_new (5, 5, TRUE);

/* --- Tworzymy przyciski kalkulatora --- */
UtworzPrzyciskiKalkulatora (tabela);
```

Pole wyświetlacza jest kontrolką etykiety, wyświetlającą tekst wyrównany do prawej strony, a więc w sposób znany z prawdziwych kalkulatorów. Wyświetlacz jest dodawany do pierwszego rzędu tabeli i rozciąga się na pięć kolumn.

```
/* --- Tworzymy wyświetlacz LCD kalkulatora --- */
etykieta = gtk_label_new ("0");
gtk_misc_set_alignment (GTK_MISC (etykieta), 1, .5);

/* --- Dodajemy etykietę do tabeli  --- */
```

```
    gtk_table_attach_defaults (GTK_TABLE (tabela), etykieta,
                               0, 4, 0, 1);
```

Następnie uwidaczniamy wszystkie kontrolki i dodajemy je do głównego okna.

```
    gtk_widget_show (etykieta);

     /* --- Uwidaczniamy tabelę i okno --- */
    gtk_container_add (GTK_CONTAINER (okno), tabela);

    gtk_widget_show (tabela);
    gtk_widget_show (okno);
```

Wreszcie wywołujemy gtk_main, aby rozpocząć przetwarzanie zdarzeń.

```
    gtk_main ();
    exit (0);
}
```

Procedura obsługi delete_event zamknie GTK+, kiedy użytkownik zakończy aplikację kalkulatora.

```
/*
 * ZamknijOknoApl
 *
 * Okno jest zamykane, wychodzimy z pętli GTK
*/
gint ZamknijOknoApl (GtkWidget *widget, gpointer data)
{
    gtk_main_quit ();

    return (FALSE);
}
```

Utwórz Przyciski Kalkulatora

Funkcja UtworzPrzyciskiKalkulatora tworzy przyciski na podstawie tablicy, opisanej we wcześniejszej części rozdziału. Każdemu tworzonemu przyciskowi przypisuje etykietę, rząd i kolumnę określoną w tablicy. Wskaźnik GtkWidget, reprezentujący nowo utworzony przycisk, jest zapisywany w tablicy. Funkcja UtworzPrzyciskiKalkulatora korzysta z funkcji UtworzPrzycisk, która tworzy jeden przycisk.

```
void UtworzPrzyciskiKalkulatora (GtkWidget *tabela)
{
```

```
    int nIndeks;

    /* --- Przechodzimy przez listę przycisków  --- */
    for (nIndeks = 0; nIndeks < nPrzyciski; nIndeks++) {

      /* --- Tworzymy przycisk --- */
      listaPrzyciskow[nIndeks].kontrolka =
            UtworzPrzycisk (tabela,
                  listaPrzyciskow[nIndeks].szEtykieta,
                  listaPrzyciskow[nIndeks].wiersz,
                  listaPrzyciskow[nIndeks].kol);
    }
}
```

Utwórz Przycisk

Funkcja UtworzPrzycisk tworzy przycisk z etykietą i umieszcza go w tablicy pakującej. Funkcja ta ustawia także procedurę obsługi zdarzenia "clicked", kliknieto_przycisk. Zauważmy, że wszystkie przyciski korzystają z tej samej funkcji zwrotnej kliknieto_przycisk. Zamiast pobierać etykietę z przycisku (w celu sprawdzenia, który przycisk spowodował zdarzenie), łatwiej jest skonfigurować funkcję zwrotną tak, aby otrzymywała tę etykietę jako parametr. Po odpowiednim ustawieniu funkcji kliknieto_ przycisk w wywołaniu gtk_signal_connect, będzie ona otrzymywała etykietę przycisku w parametrze typu gpointer.

```
/*
 * UtworzPrzycisk
 *
 * Tworzy przycisk, przypisuje procedury obsługi zdarzenia, i
 * umieszcza przycisk we właściwej komórce tabeli pakującej
*/
GtkWidget *UtworzPrzycisk (GtkWidget *tabela, char *szEtykieta,
                  ➥ int wiersz, int kolumna)
{
    GtkWidget *przycisk;

    /* --- tworzymy przycisk --- */
    przycisk = gtk_button_new_with_label (szEtykieta);

    /* --- musimy wiedzieć, kiedy przycisk zostanie kliknięty --- */
    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (kliknieto_przycisk), szEtykieta);
```

```
    /* --- umieszczamy przycisk we właściwej komórce tabeli --- */
    gtk_table_attach (GTK_TABLE (tabela), przycisk,
                      kolumna, kolumna+1,
                      wiersz, wiersz + 1,
                     GTK_FILL | GTK_EXPAND,
                     GTK_FILL | GTK_EXPAND,
                     5, 5);

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

    /* --- Zwracamy przycisk --- */
    return (przycisk);
}
```

Procedura obsługi, która będzie przetwarzać zdarzenia pochodzące od przycisków i klawiszy, musi określić, który przycisk wciśnięto i na tej podstawie odpowiednio uaktualnić wyświetlacz kalkulatora. W przypadku kliknięć myszy zdarzenie otrzymuje przycisk, na którym kliknięto. Natomiast w przypadku naciśnięcia klawisza (key_press_event) zdarzenie prawdopodobnie otrzyma inny przycisk, niż chciał nacisnąć użytkownik. Jeśli na przykład ognisko (*focus*) spoczywa na przycisku „3", a użytkownik naciśnie klawisz „5", wówczas to przycisk „3" otrzyma sygnał o naciśnięciu klawisza „5". Obsługa klawiszy powinna być podobna do obsługi sygnału clicked, gdzie sygnał otrzymuje kliknięty przycisk (po naciśnięciu klawisza „5" odpowiedni sygnał powinien otrzymać przycisk „5"). Co więcej, jeśli użytkownik korzysta z klawiatury, ognisko powinno przesuwać się na przycisk odpowiadający wciśniętemu klawiszowi; w omawianym przykładzie ognisko powinno przesunąć się na przycisk „5", ponieważ wciśnięto klawisz „5". Jest to dodatkowe, wizualne potwierdzenie czynności użytkownika, kwestia raczej estetyczna, niż programistyczna.

Zamiast przypisywać poszczególnym przyciskom funkcję zwrotną dla sygnału key_press_event, możemy przypisać ją całemu oknu aplikacji, do którego trafiają wszystkie sygnały, które nie zostały obsłużone przez kontrolkę z ogniskiem. Funkcja zwrotna dla okna aplikacji, obsługująca sygnał key_press_event, wyśle sygnał clicked do właściwego przycisku.

Zdarzenia kliknięcia myszy i naciśnięcia klawisza prowadzą do podjęcia przez program tych samych czynności. Procedura obsługi zdarzenia key_press_event może wykorzystać ten fakt, wywołując te same funkcje, które używane są podczas przetwarzania sygnału clicked.

Funkcja zwrotna dla key_press_event jest niewielka, ponieważ musi ona tylko określić, który przycisk należy kliknąć, na podstawie naciśniętego klawisza. Przegląda ona przyciski w tablicy i próbuje dopasować klawisz do tekstu, wyświetlanego na etykiecie przycisku. Porównywany jest tylko pierwszy znak etykiety, więc nie da się użyć tej procedury dla bardziej skomplikowanych operacji kalkulatora (na przykład 1/x), które trudno jest wyrazić pojedynczym wciśnięciem klawisza. Operuje ona tylko na częściej używanych symbolach, jak cyfry i proste operatory (+-/*). Po odnalezieniu właściwego przycisku funkcja zwrotna przesuwa na niego ognisko w celu wizualnego potwierdzenia operacji, a następnie wywołuje funkcję gtk_button_clicked, aby wygenerować sygnał clicked dla tego przycisku.

```
/*
 * wcisnieto_klawisz
 *
 * Obsługuje sygnał "key_press_event"
 *
 * Funkcja poszukuje odpowiednika klawisza w strukturze
 * danych kalkulatora i (jeśli go znajdzie) klika w imieniu
 * użytkownika właściwy przycisk. Zmniejsza to nasz program,
 * ponieważ musimy obsłużyć tylko zdarzenia "clicked".
*/
void wcisnieto_klawisz (GtkWidget *kontrolka,
                GdkEventKey *zdarzenie,
                gpointer dane)
{
   int nIndeks;

   /* --- przeglądamy przyciski --- */
   for (nIndeks = 0; nIndeks < nPrzyciski; nIndeks++) {

     /* --- Jeśli wciśnięty klawisz odpowiada pierwszemu --- */
     /* --- znakowi etykiety przycisku ORAZ długość etykiety --- */
     /* --- jest równa jeden --- */
     if (zdarzenie->keyval ==
     listaPrzyciskow[nIndeks].szEtykieta[0] &&
     listaPrzyciskow[nIndeks].szEtykieta[1] == (char) 0) {

       /* --- Przesuwamy ognisko na ten przycisk --- */
       gtk_widget_grab_focus (listaPrzyciskow[nIndeks].kontrolka);

       /* --- Symulujemy kliknięcie przycisku --- */
```

```
            gtk_button_pressed (GTK_BUTTON (
                        listaPrzyciskow[nIndeks].kontrolka));
            return;
          }
        }
      }
```

Jeśli użytkownik kliknie przycisk, albo naciśnie klawisz, zostanie wywołana funkcja obsługi zdarzenia kliknieto_przycisk. Funkcja ta przechowuje aktualny stan kalkulatora i przeprowadza żądane operacje, albo też dodaje cyfry do wyświetlacza przy pomocy funkcji gtk_label_set. Naturalnie, funkcja kliknieto_przycisk mogłaby obsługiwać pierwszeństwo operatorów, nawiasy, dalsze operacje (sin, cos, tan), a także konwersje na format szesnastkowy i binarny, ale pozostawimy to inwencji Czytelnika.

Funkcja kliknieto_przycisk rozróżnia dwa podstawowe przypadki. Użytkownik albo wprowadza cyfrę (od 0 do 9) albo też żąda przeprowadzenia operacji. Jeśli użytkownik wciska cyfrę, zazwyczaj oznacza to, że należy po prostu dodać ją do wyświetlacza. Jeśli na wyświetlaczu znajduje się na przykład liczba „45", a użytkownik naciśnie „2", wówczas wyświetlacz powinien pokazać „452". Istnieje wyjątek od tej reguły, jeśli poprzednio wprowadzonym znakiem był operator binarny (+-/*). Kiedy użytkownik kliknie operator binarny, wówczas wyświetlacz będzie nadal pokazywał liczbę, ale nie należy dodawać do niej nowej cyfry. Nowa cyfra jest częścią drugiego operandu i powinna zastąpić poprzednią liczbę.

Z drugim przypadkiem mamy do czynienia wtedy, kiedy użytkownik wprowadza operator. W takim liczba na wyświetlaczu musi zostać użyta łącznie z liczbą, zapamiętaną wcześniej w buforze. Załóżmy, że użytkownik wpisał „235+111". Do tej pory kalkulator nie przeprowadził żadnych obliczeń. Kiedy użytkownik wprowadzi znak „=", wówczas kalkulator sprawdza poprzedni operator („+"), zapamiętaną w buforze liczbę („235") oraz liczbę na wyświetlaczu. Teraz musi wykonać dodawanie „235" i „111". Taka sama sytuacja miałaby miejsce, gdyby użytkownik nacisnął „+", zamiast „=", po wprowadzeniu „235+111". Różnica polega na tym, że drugi znak „+" wprowadziłby kalkulator w tryb dodawania następnej liczby.

```
/*
 * kliknieto_przycisk
 *
 * kontrolka - kliknięty przycisk
 * dane - etykieta przycisku
 *
```

```
 * Kliknięto przycisk, obsługujemy go
*/
void kliknieto_przycisk (GtkWidget *kontrolka, gpointer dane)
{
   char zn = *((char *) dane);
   char *lancuch;

   /* --- Pobieramy etykietę przycisku --- */
   lancuch = (char *) dane;

   /* --- Wprowadzanie cyfry... --- */
   if (ZnakZmiennoprzecinkowy (zn) && strlen (lancuch) == 1) {

      ObsluzCyfre (lancuch, zn);

   } else {

      /* --- Czyszczenie --- */
      if (strcmp (lancuch, "CE") == 0) {
         gtk_label_set (GTK_LABEL (etykieta), "0");
         return;

      /* --- DUŻE czyszczenie? --- */
      } else if (strcmp (lancuch, "C") == 0) {
         poprzPolecenie = (char) 0;
         ostatniZnak = (char) 0;
         gtk_label_set (GTK_LABEL (etykieta), "0");
         return;

      } else {

         /* --- Może to operator unarny? --- */
         MozeOperatorUnarny (lancuch);
      }

      /* --- Sprawdzamy, czy jest do przeprowadzenia --- */
      /* --- jakaś operacja binarna  --- */
      ObsluzOperacjeBinarna ();

      poprzPolecenie = zn;
   }
   ostatniZnak = zn;
}
```

Kod, który obsługuje cyfry, dodaje je na końcu wyświetlacza, o ile poprzednio wprowadzony klawisz nie był poleceniem. W takim przypadku cyfra staje się pierwszą w wyświetlaczu.

```
/*
 * ObsluzCyfre
 *
 * Cyfra button was pressed, deal with it.  How it
 * is dealt with depends on the situation.
*/
void ObsluzCyfre (char *lancuch, char zn)
{
   char *tekstEtykiety;
   char bufor[BUF_SIZE];
   int  dlug;

   /* --- Jeśli poprzednio wprowadzono polecenie... --- */
   if (Polecenie (ostatniZnak)) {

      /* --- czyścimy wyświetlacz --- */
      gtk_label_set (GTK_LABEL (etykieta), "");

      /* --- jeśli użytkownik wykonał obliczenie --- */
      if (ostatniZnak == '=') {

         /* --- Zerujemy polecenie --- */
         ostatniZnak = (char) 0;
         poprzPolecenie = (char) 0;
      }
   }

   /* --- Pobieramy tekst w etykiecie do bufora --- */
   gtk_label_get (GTK_LABEL (etykieta), &tekstEtykiety);
   strcpy (bufor, tekstEtykiety);

   /* --- Dodajemy do niego nowy zmal --- */
   dlug = strlen (bufor);
   bufor[dlug] = (gchar) zn;
   bufor[dlug+1] = (gchar) 0;

     /* --- Obcinamy wiodące zera. --- */
   ObetnijWiodaceZera (bufor);

   /* --- Dodajemy znak do wyświetlacza --- */
```

```
    gtk_label_set (GTK_LABEL (etykieta), (char *) bufor);
}
```

Kalkulator obsługuje dwa typy poleceń: takie, które przyjmują dwie liczby w celu przeprowadzenia operacji, oraz takie, które przyjmują tylko jedną. Funkcja MozeOperatorUnarny sprawdza, czy polecenie jest operatorem unarnym, a jeśli tak, wówczas próbuje przeprowadzić unarną operację.

```
/*
 * MozeOperatorUnarny
 *
 * lancuch - etykieta na przycisku, która opisuje operację
 *
 * Sprawdza, czy użytkownik nacisnął przycisk z operatorem
 * unarnym, jak %, pierw, 1/x etc., który należy obsłużyć
 * natychmiast
*/
void MozeOperatorUnarny (char *lancuch)
{
    char *tekstEtykiety;
    char bufor[BUF_SIZE];
    float num2;

    /* --- Pobieramy liczbę z wyświetlacza --- */
    gtk_label_get (GTK_LABEL (etykieta), &tekstEtykiety);
    num2 = atof (tekstEtykiety);

    /* --- Procent? --- */
    if (strcmp (lancuch, "%") == 0) {
        num2 = num2 / 100;

    /* --- Może 1/x? --- */
    } else if (strcmp (lancuch, "1/x") == 0) {

        /* --- Nie można dzielić przez 0 --- */
        if (num2 == 0) {
            return;
        }
        num2 = 1 / num2;

    /* --- Obliczanie pierwiastka --- */
    } else if (strcmp (lancuch, "pierw") == 0) {
        num2 = sqrt ((double) num2);
```

```
    /* --- Obliczanie kwadratu --- */
    } else if (strcmp (lancuch, "x^2") == 0) {
       num2 = num2 * num2;
    }

    /* --- Umieszczamy obliczoną liczbę w wyświetlaczu --- */
    sprintf (bufor, "%f", (float) num2);
    ObetnijKoncoweZera (bufor);
    ObetnijWiodaceZera (bufor);
    gtk_label_set (GTK_LABEL (etykieta), bufor);
}
```

Funkcja ObsluzOperacjeBinarna sprawdza, czy należy przeprowadzić operację binarną. Globalna zmienna num1 przechowuje pierwszą liczbę, która zostanie użyta w obliczeniach. Druga liczba jest pobierana z wyświetlacza kalkulatora. Kiedy obliczenia dobiegną końca, rezultat jest zapamiętywany w globalnej zmiennej num1, ponieważ może stać się operandem w kolejnej operacji binarnej.

```
void ObsluzOperacjeBinarna ()
{
   char bufor[BUF_SIZE];
   char *tekstEtykiety;
   float num2;

   /* --- Pobieramy liczbę z wyświetlacza --- */
   gtk_label_get (GTK_LABEL (etykieta), &tekstEtykiety);
   num2 = atof (tekstEtykiety);

   /* --- Przeprowadzamy obliczenia na podstawie ostatnio --- */
   /* --- wprowadzonego operatora --- */
   switch (poprzPolecenie) {
      case '+':
         num1 = num1 + num2;
         break;

      case '-':
         num1 = num1 - num2;
         break;

      case '*':
         num1 = num1 * num2;
         break;
```

```
        case '/':
           num1 = num1 / num2;
           break;

        case '=':
           num1 = num2;
           break;

        default:
           num1 = num2;
           break;
     }

     /* --- Umieszczamy obliczoną liczbę w wyświetlaczu --- */
     sprintf (bufor, "%f", (float) num1);
     ObetnijKoncoweZera (bufor);
     ObetnijWiodaceZera (bufor);
     gtk_label_set (GTK_LABEL (etykieta), bufor);
  }
```

Pozostałe funkcje są funkcjami pomocniczymi. Sprawiają, że program staje się czytelniejszy, ponieważ ich nazwy dokumentują przeprowadzane przez niego operacje. Na przykład funkcja ZnakZmiennoprzecinkowy sprawdza, czy wprowadzany znak może występować w liczbie zmiennoprzecinkowej (0-9 albo kropka „.”). Łatwiej czyta się linię kodu w rodzaju:

```
if (ZnakZmiennoprzecinkowy (zn)) {
```

niż linię:

```
if (isdigit (zn) || zn == ".") {
```

```
/*
 * ObetnijKoncoweZera
 *
 * Pozbywamy się końcowych zer.
 * Funkcja przyjmuje łańcuch i obcina końcowe zera
*/
void ObetnijKoncoweZera (char *szCyfry)
{
   int nIndeks;
   int bKropka = FALSE;
   int nPoz = -1;
```

```
        /* --- Przeglądamy łańcuch w pętli --- */
    for (nIndeks = 0; nIndeks < strlen (szCyfry); nIndeks++) {

        /* --- Czy to kropka dziesiętna? --- */
        if (szCyfry[nIndeks] == '.') {
            bKropka = TRUE;
        }

        /* --- Jeśli jesteśmy po prawej stronie kropki... --- */
        if (bKropka) {

            /* --- Zero? Hmm... czy już do końca? --- */
            if (szCyfry[nIndeks] == '0') {

                /* --- Jeśli nie mamy zapamiętanej pozycji... --- */
                if (nPoz < 0) {

                    /* --- ...to zapamiętujemy ją teraz --- */
                    nPoz = nIndeks;
                }
            } else {

                /* --- Usuwamy zapamiętaną pozycję. To nie ta. --- */
                nPoz = -1;
            }
        }
    }

    /* --- Przycinamy łańcuch --- */
    if (nPoz > 0) {
        szCyfry[nPoz] = (char) 0;
    }
}

/*
 * ObetnijWiodaceZera
 *
 * Usuwa wiodące zera.
 *
 * Przekształca liczby w rodzaju "0000012" na "12"
*/
void ObetnijWiodaceZera (char *szCyfry)
{
```

```
    int nPoz;

    if (szCyfry == NULL) return;

    /* --- dopóki mamy kombinację: zero na przedzie... --- */
    for (nPoz = 0; (szCyfry[nPoz] && szCyfry[nPoz] == '0'); nPoz++) {

        /* --- ...a następny znak również jest cyfrą... --- */
        if (isdigit (szCyfry[nPoz+1])) {

            /* --- ...zastępujemy zero spacją --- */
            szCyfry[nPoz] = ' ';
        }
    }
}

/*
 * Polecenie
 *
 * Zwraca wartość TRUE, jeśli znak jest poleceniem
 * operującym na dwóch liczbach.
*/
int Polecenie (char zn)
{
    switch (zn) {
        case '+':
        case '-':
        case '/':
        case '*':
        case '=':
            return (TRUE);
    }
    return (FALSE);
}

/*
 * ZnakZmiennoprzecinkowy
 *
 * Zwraca TRUE, jeśli znak jest jednym z [0123456789.]
*/
int ZnakZmiennoprzecinkowy (char zn)
{
```

```
    return (isdigit (zn) || zn == '.');
}
```

Podsumowanie

Napisanie prostego kalkulatora z wykorzystaniem kontrolek przycisków i etykiety jest dość łatwe. Dorzucamy do kontrolek kilka funkcji obsługi zdarzenia i otrzymujemy gotową aplikację. Nic nie stoi na przeszkodzie, aby przystąpić do pisania nieco większego programu.