

Mirosław J. Kubiak

ZŁOTE
MYŚLI

Programuję w Delphi i C++ Builder

II część

dla zaawansowanych

**Jak szybko nauczyć
się programowania
w dwóch różnych językach**

Ten ebook zawiera darmowy fragment publikacji "[Programuję w Delphi i C++ Builder - cz.2](#)"

Darmowa publikacja dostarczona przez
www.darmowe-ebooki.pl

Copyright by Złote Myśli & Mirosław J. Kubiak, rok 2008

Autor: Mirosław J. Kubiak

Tytuł: Programuję w Delphi i C++ Builder - cz.2

Data: 02.12.2011

Złote Myśli Sp. z o.o.

ul. Toszecka 102

44-117 Gliwice

www.zlotemysli.pl

email: kontakt@zlotemysli.pl

Niniejsza publikacja może być kopiowana, oraz dowolnie rozprowadzana tylko i wyłącznie w formie dostarczonej przez Wydawcę. Zabronione są jakiegokolwiek zmiany w zawartości publikacji bez pisemnej zgody Wydawcy. Zabrania się jej odsprzedaży, zgodnie z regulaminem Wydawnictwa Złote Myśli.

Autor oraz Wydawnictwo Złote Myśli dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo Złote Myśli nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wszelkie prawa zastrzeżone.

All rights reserved.

SPIS TREŚCI

<u>Rozdział 8. Tablice</u>	5
<u>Deklarowanie tablic</u>	5
<u>Dostęp do elementów tablicy</u>	11
<u>Sito Eratostenesa</u>	14
<u>Tablice dwuwymiarowe</u>	17
<u>Sortowanie bąbelkowe</u>	23
<u>Tablice przechowują również teksty</u>	28
<u>Warto zapamiętać</u>	42
<u>Rozdział 9. Podprogramy</u>	43
<u>Co to są podprogramy</u>	43
<u>Definiujemy funkcje i procedury w języku Delphi</u>	43
<u>Definiujemy funkcje w języku C++ Builder</u>	48
<u>Programy mogą przekazywać podprogramom informacje</u>	52
<u>Przekazywanie parametrów przez wartość</u>	54
<u>Zmienne globalne i lokalne</u>	58
<u>Przekazujemy funkcjom i procedurom tablice</u>	59
<u>Rekurencja</u>	68
<u>Warto zapamiętać</u>	75
<u>Rozdział 10. Elementy grafiki</u>	76
<u>Wprowadzenie</u>	76
<u>Program Przykładowa grafika</u>	85
<u>Graficzny przykład programu rekurencyjnego</u>	92
<u>Warto zapamiętać</u>	110
<u>Rozdział 11. Przechowywanie informacji w rekordach i w strukturach</u>	111
<u>Rekordy i struktury</u>	111
<u>Warto zapamiętać</u>	123
<u>Rozdział 11. Elementy programowania obiektowego</u>	124
<u>Wprowadzenie</u>	124
<u>Hermetyzacja danych</u>	147
<u>Dziedziczenie</u>	147
<u>Polimorfizm</u>	158
<u>Rozdział 13. Operacje wejścia/wyjścia – część II. Pliki</u>	160
<u>Wprowadzenie</u>	160
<u>C++ Builder</u>	162
<u>Pliki tekstowe</u>	163
<u>Zapisywanie rekordów i struktur do pliku</u>	182
<u>C++ Builder</u>	191
<u>Warto zapamiętać</u>	200
<u>Rozdział 14. Wskaźniki</u>	202
<u>Wprowadzenie</u>	202
<u>Wskaźniki i tablice</u>	207
<u>Programy mogą przekazywać podprogramom informacje – ciąg dalszy</u>	213
<u>Przekazywanie parametrów przez wskaźnik</u>	214
<u>Przekazywanie parametrów przez referencje</u>	217
<u>Warto zapamiętać</u>	219

<u>Rozdział 15. Zmienne dynamiczne</u>	220
<u>Wprowadzenie</u>	220
<u>Zmienne dynamiczne do tablic</u>	225
<u>Klasy TStringList i TStringList</u>	228
<u>Warto zapamiętać</u>	230
<u>Rozdział 16. Algorytmy numeryczne</u>	232
<u>Obliczanie sumy szeregu</u>	232
<u>Wyznaczenie miejsca zerowego funkcji metodą Newtona</u>	237
<u>Wyznaczanie miejsca zerowego funkcji</u>	244
<u>Obliczanie całki metodą prostokątów</u>	249
<u>Dodatek</u>	256
<u>D1. Formatowanie łańcuchów tekstowych</u>	256
<u>D2. Wybrane systemowe procedury konwersji typu</u>	258
<u>D3. Standardowe procedury obsługujące pliki (Delphi)</u>	260
<u>D4. Wyświetlanie komunikatów</u>	263
<u>D5. Wartości parametru Flags dotyczące liczby i rodzaju przycisków</u>	264
<u>D6. Grafika w Delphi i w C++ Builder</u>	268
<u>Bibliografia</u>	272

Rozdział 8. Tablice

W tym rozdziale dowiemy się, w jaki sposób deklarujemy tablice jedno- i dwuwymiarowe, na czym polega sortowanie bąbelkowe oraz o tym, że tablice przechowują nie tylko liczby, ale również teksty.

Deklarowanie tablic

Tablica jest to struktura danych, która umożliwia przechowywanie w sposób zorganizowany wielu zmiennych tego samego typu (całkowitego, rzeczywistego itd.). Aby stworzyć taką strukturę musimy dokonać deklaracji tablicy. W deklaracji tablicy musimy określić typ wartości, jaki ma przechowywać tablica, a także liczbę jej elementów. Tablice mogą być jednowymiarowe, dwuwymiarowe itd.

Delphi

Oto ogólna postać deklarowania w języku Delphi tablicy jednowymiarowej i związanej z nią zmiennej.

type

dentyfikator_tablicy = array[rozmiar_tablicy] of typ

var

nazwa_zmiennej : identyfikator_tablicy;

A oto przykład zadeklarowania tablicy jednowymiarowej o nazwie **dane** typu całkowitego, zawierającej 10 elementów.

type

```
tablica = array[1..10] of integer;
```

var

```
dane : tablica;
```

możliwy jest również inny poprawny zapis:

var

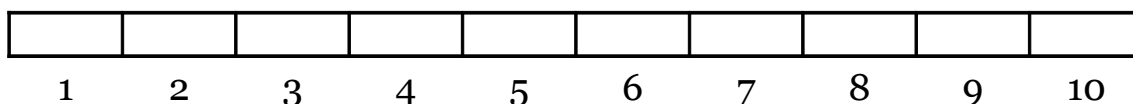
```
dane : array[1..10] of integer;
```

Dostęp do elementów tablicy jest realizowany za pośrednictwem indeksu, który wskazuje dany element. Dla deklaracji tablicy w języku Delphi zawartej poniżej:

var

```
dane : array[1..10] of integer;
```

pierwszy element tablicy **dane** ma indeks 1, drugi element dostępny jest przez indeks 2 itd. Ostatni element tablicy ma indeks równy wymiarowi tablicy, czyli 10, co zilustrowano na rysunku poniżej.



Oto prosty przykład programu ilustrujący posługiwanie się tablicą jednowymiarową.

Program (Tablica1)

Formularz

Na formularzu wykonujemy następujące czynności:

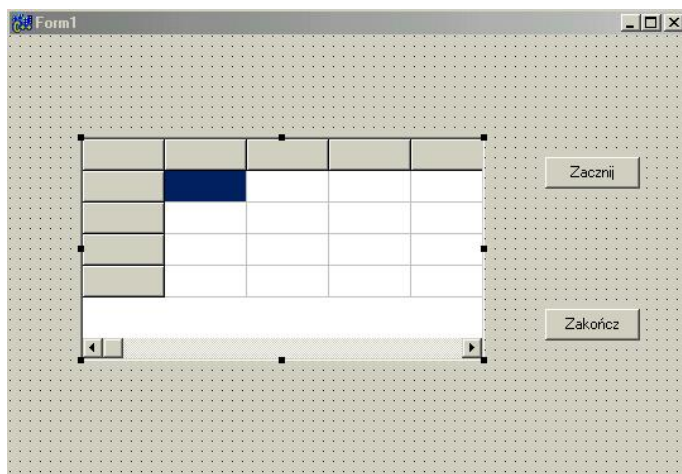
1. umieszczamy dwa przyciski **Button**, jeden u góry z prawej strony (**Button1**) i jeden na dole również z prawej strony (**Button2**),
2. następnie klikamy myszą raz na **Button1** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button1** na **Zacznij**,
3. klikamy myszą raz na **Button2** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button2** na **Zakończ**,
4. z lewej górnej strony formularza umieszczamy komponent **StringGrid**,
5. dwukrotnie klikamy na formularzu przycisk **Zakończ** i podpinamy związaną z nim procedurę **procedure TForm1.Button2Click(Sender: TObject);**

begin

Close;

end;

6. następnie dwukrotnie klikamy na formularzu przycisk **Zacznij** i podpinamy związaną z nim procedurę **procedure TForm1.Button1Click(Sender: TObject)**, która została opisana w programie poniżej.



Rys. 8.1. Formularz skonstruowany dla programu *Tablica1*, na którym zamieszczono komponent *StringGrid*.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, StdCtrls, Grids;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    Button2: TButton;  
    StringGrid1: TStringGrid;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
const
  n = 10;

type
  tablica = array[1..n] of integer;
var
  i : integer;
  dane : tablica;

begin
  StringGrid1.ColCount:=2; // ustalenie ilosci kolumn
  StringGrid1.RowCount:=n+1; //ustalenie ilosci wierszy
  StringGrid1.Cells[0,0]='Indeks tabl.';
  StringGrid1.Cells[1,0]='Wart. tabl.';

  for i:=1 to n do
    StringGrid1.Cells[0,i]:=IntToStr(i); //opis wierszy

  for i:=1 to n do
    begin
      dane[i]:=i;
      StringGrid1.Cells[1,i]:=IntToStr(dane[i]);
    end;
  end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close();
end;

end.
```

C++ Builder

Oto ogólna postać zadeklarowania w języku w C++ Builder tablicy jednowymiarowej i związanej z nią zmiennej.

typ_tablicy nazwa_tablicy [rozmiar_tablicy]

A oto przykład zadeklarowania tablicy jednowymiarowej o nazwie **dane** typu całkowitego, zawierającej 10 elementów:

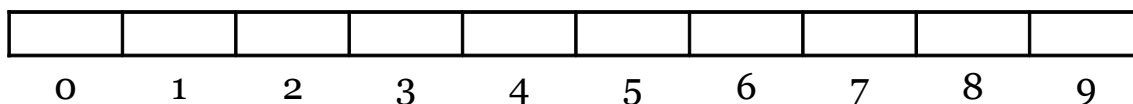
```
int dane [10];
```

Dostęp do elementów tablicy

Dostęp do elementów tablicy jest realizowany za pośrednictwem indeksu, który wskazuje dany element. Dla deklaracji tablicy

```
int dane [10];
```

aby uzyskać dostęp do pierwszego elementu tablicy **dane**, powinniśmy podać indeks 0¹, drugi element dostępny jest przez indeks 1 itd. Ostatni element tablicy ma indeks równy rozmiarowi tablicy minus 1, czyli 9, co zilustrowano na rysunku poniżej.



Oto prosty przykład ilustrujący posługiwanie się tablicą jednowymiarową.

¹Indeks pierwszego elementu tablicy w języku C++ wynosi zawsze 0.

Program (Tablica1)

Formularz

Na formularzu wykonujemy następujące czynności:

1. umieszczamy dwa przyciski **Button**, jeden u góry z prawej strony (**Button1**) i jeden na dole również z prawej strony (**Button2**),
2. następnie klikamy myszą raz na **Button1** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button1** na **Zaczniij**,
3. klikamy myszą raz na **Button2** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button2** na **Zakończ**,
4. z lewej górnej strony formularza umieszczamy komponent **StringGrid**,
5. dwukrotnie klikamy na formularzu przycisk **Zakończ** i podpinamy związaną z nim funkcję **void __fastcall TForm1::Button2Click(TObject *Sender)**

```
{  
  
    Close();  
  
}
```
6. następnie dwukrotnie klikamy na formularzu przycisk **Zaczniij** i podpinamy związaną z nim funkcję **void __fastcall TForm1::Button1Click(TObject *Sender)**, która została opisana w programie poniżej.

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  

```

```
//-----  
void __fastcall TForm1::Button2Click(TObject *Sender)  
{  
  Close();  
}  
//-----
```

Sito Eratostenesa

Okolo roku 200 p.n.e. matematyk grecki Eratostenes podal algorytm na znajdowanie liczb pierwszych. Liczby pierwsze to sa liczby naturalne wieksze od 1, ktore dzielą się przez siebie i przez 1. Nazwa tego najstarszego algorytmu na znajdowanie liczb pierwszych pochodzi od sposobu, w jaki te liczby sa znajdowane. Wszystkie liczby po kolei przesiewa się, usuwając z spóród nich wszystkie wielokrotności danej liczby. Zilustrujemy to przykladem, znajdując za pomoca sita Eratostenesa wszystkie liczby pierwsze z zakresu od 2 do 30, ktore umieścimy w tabeli ponizej.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	

Liczba 2 jest liczbą pierwszą, pozostałe znajdujemy po usunięciu z tabeli wielokrotności liczby 2 (gdyż nie są to liczby pierwsze) otrzymując:

2	3	*	5	*	7	*	9	*	11	*	13	*	15	*
17	*	19	*	21	*	23	*	25	*	27	*	29	*	

Kolejny etap polega na usunięciu z tabeli po liczbie 3 (przyjmuje się, że jest to liczba pierwsza) wielokrotności liczby 3. Rezultat tego działania znajduje się poniżej:

2	3	*	5	*	7	*	*	*	11	*	13	*	*	*
17	*	19	*	*	*	23	*	25	*	*	*	29	*	

Z pozostałych teraz liczb kolejną po 2 i 3 jest liczba 5, którą pozostawia się wykreślając wszystkie liczby podzielne przez 5:

2	3	*	5	*	7	*	*	*	11	*	13	*	*	*
17	*	19	*	*	*	23	*	*	*	*	*	29	*	

Kontynuując to wykreślanie dojdziemy do sytuacji, kiedy zostaną wykreślone wszystkie liczby, które nie są pierwsze i pozostaną tylko liczby pierwsze.

W tym momencie możemy zakończyć nasze poszukiwania liczb pierwszych pamiętając, że kolejne wykreślenia należy powtarzać, nie dalej jak do liczby będącej zaokrąglonym w dół pierwiastkiem kwadratowym ze zmiennej **zakres**. W naszym przykładzie jest to: $\text{sqrt}(30)=5.477\dots$, po zaokrągleniu w dół² otrzymujemy liczbę 5. W tabeli pozostały już tylko liczby pierwsze, które wyświetlamy na ekranie.

Górny zakres, dla którego chcemy odnaleźć liczby pierwsze wyznaczony jest tylko rozmiarem tablicy. W naszym programie jest to:

²W języku C++ Builder realizuje to matematyczna funkcja **floor()**, która znajduje się w pliku nagłówkowym **math.h**.

```
int tablica[10000];
```

Chcąc zmienić ten zakres, należy zmienić rozmiar tablicy.

C++ Builder (Sito)

```
//-----  
#include <vcl.h>  
#include <math.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----  
  
void __fastcall TForm1::Button2Click(TObject *Sender)  
{  
    Close();  
}  
//-----  
  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    int i,j,zakres,zm_pom;  
    int tablica[10000];  
  
    zakres=StrToInt(InputBox("Podaj gorny zakres liczb
```

```
pierwszych","Maksymalny gorny zakres wynosi 9999",""));  
  
zm_pom = floor(sqrt(zakres));  
  
for (i=1; i<=zakres; i++)  
  tablica[i]=i;  
  for (i=2; i<=zm_pom; i++)  
    if (tablica[i]!=0)  
      for (j=i+1; j<=zakres; j++)  
        if (j%i==0) tablica[j]=0;  
  
ListBox1->Items->Clear();  
ListBox1->Items->Add("Liczby pierwsze z zakresu od 1 do  
"+IntToStr(zakres)+" to");  
  
for (i=2; i<=zakres; i++)  
  if (tablica[i]!=0) ListBox1->Items->Add(IntToStr(i));  
}  
//-----
```

Tablice dwuwymiarowe

Tablice dwuwymiarowe deklarujemy analogicznie jak tablice jednowymiarowe. Ilustrują to poniższe przykłady. Tablice wielowymiarowe deklarujemy analogicznie.

Delphi

A oto program napisany w języku Delphi, który w zadeklarowanej tablicy dwuwymiarowej 10x10 umieszcza na przekątnej jedynek, a poza przekątną zera. W programie skorzystaliśmy z komponentu **StringGrid**.

Program (**Tablica2**) – formularz jak na Rys. 8.1.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, StdCtrls, Grids;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    Button2: TButton;  
    StringGrid1: TStringGrid;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{ $R *.dfm }  
  
procedure TForm1.Button1Click(Sender: TObject);  
  
  const  
    n = 10;  
  
  type  
    macierz = array[1..n, 1..n] of integer;
```

```
var
  i, j : integer;
  tablica : macierz;

begin
  StringGrid1.ColCount:=n+1; //ustalenie ilosci kolumn
  tabeli
  StringGrid1.RowCount:=n+1; //ustalenie ilosci wierszy
  tabeli
  StringGrid1.DefaultColWidth:=25; //ustalenie wysokosci
  komorki
  StringGrid1.DefaultRowHeight:=25; //ustalenie
  szerokosci komorki

  for i:=1 to n do
    begin
      StringGrid1.Cells[i,0]:=IntToStr(i);
      StringGrid1.Cells[0,i]:=IntToStr(i);
    end;

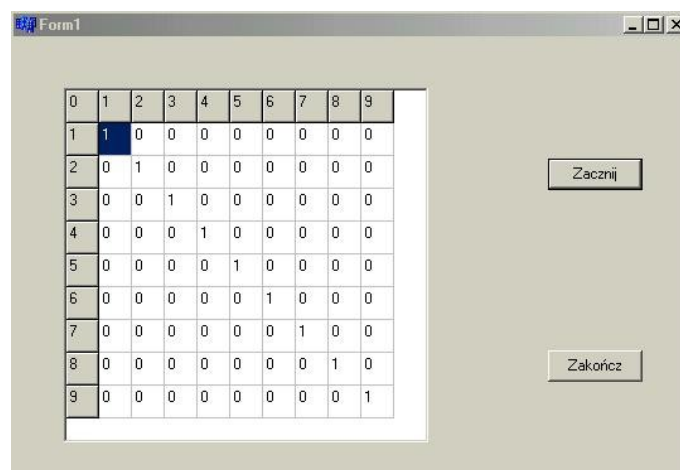
  {wpisywanie do tablicy 1 na przekatnej, a 0 poza
  przekatna}
  for i:=1 to n do
    begin
      for j:=1 to n do
        begin
          if i = j then tablica[i,j]:=1
          else tablica[i,j]:=0
        end; {j}
      end; {i}
    end;
  {Koniec wpisywania}

  {Wydruk zawartosci tablicy}
  for i:=1 to n do
    begin
      for j:=1 to n do
        begin
          StringGrid1.Cells[i,j]:= IntToStr(tablica[i,j]);
```

```
end; {j}
end; {i}
{Koniec wydruku}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close();
end;

end.
```



Rys. 8.2. Rezultat działania programu *Tablica2* napisanego w języku Delphi.

Zachęcamy Czytelnika, aby dodatkowo wzbogacił program o obliczanie sumy elementów znajdujących się na przekątnej.

C++ Builder

A oto program napisany w języku C++ Builder, który w zadeklarowanej tablicy dwuwymiarowej 10x10 umieszcza na przekątnej jedynek, a poza przekątną zera. W programie skorzystaliśmy z komponentu **StringGrid**.

Język C++ Builder (**Tablica2**) – formularz jak na Rys. 8.2.

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----  
  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    const n=11;  
    int i, j, suma;  
    int macierz[11][11];  
  
    StringGrid1->ColCount=n; //ustalenie ilosci kolumn  
    StringGrid1->RowCount=n; //ustalenie ilosci wierszy  
    StringGrid1->DefaultColWidth=25; //ustalenie szerokosci  
komorki  
    StringGrid1->DefaultRowHeight=25; //ustalenie  
szerokosci komorki  
  
    for (i=0; i<=n; i++) //opis tablicy  
    {  
        StringGrid1->Cells[i+1][0]=IntToStr(i);  
        StringGrid1->Cells[0][i+1]=IntToStr(i);  
    }  
}
```

```
//wpisywanie 1 na przekatnej, a poza przekatna zero
for (i=0; i<n; i++)
{
for (j=0; j<n; j++)
{
if (i==j)
macierz[i][j]=1; // wpisanie 1 na przekatnej
else
macierz[i][j]=0; // wpisanie 0 poza przekatna
}
} //koniec wpisywania

//wyswietlanie tablicy
for (i=0; i<n; i++)
{
for (j=0; j<n; j++)
{
StringGrid1->Cells[i+1][j+1]=IntToStr(macierz[i][j]);
}
}
}
//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Close();
}
//-----
```

Zachęcamy Czytelnika, aby dodatkowo wzbogacił program o obliczanie sumy elementów znajdujących się na przekątnej.

Sortowanie bąbelkowe

Następny program, który wykorzystuje tablice nazywa się **Bąbelki**. Metoda sortowania bąbelkowego opiera się na zasadzie porównywania i zamiany par sąsiadujących ze sobą liczb, do chwili ustawienia ich w kolejności od najmniejszej do największej. Wynika to stąd, że warunkiem uporządkowania jest jednoczesne udostępnienie wszystkich liczb (w programie jest ich 6). Jeśli potraktujemy zadane liczby jako element „pionowej” tablicy jednowymiarowej, to przy pewnej dozie wyobraźni możemy je uważać za znajdujące się w naczyniu z wodą bąbelki o „wadze” proporcjonalnej do ich wielkości. Rezultatem każdego przejścia przez tablicę będzie wypchnięcie bąbelka na poziom odpowiadający jego „wadze”.

Tabela 8.1. Przebieg sortowania bąbelkowego po kolejnych przejściach dla 6 ustalonych liczb

Ustawienie początkowe	Pierwsze przejście	Drugie przejście	Trzecie przejście	Czwarte przejście	Piąte przejście
574	8	8	8	8	8
303	574	23	23	23	23
34	303	574	34	34	34
125	34	303	574	125	125
8	125	34	303	574	303
23	23	125	125	303	574

A oto program w Delphi, który zawiera algorytm sortowania bąbelkowego.

Delphi (Babelki) – formularz jak na Rys. 8.1.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, Grids, StdCtrls;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    StringGrid1: TStringGrid;  
    Button2: TButton;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{$R *.dfm}  
  
procedure TForm1.Button1Click(Sender: TObject);  
  var  
    liczby : array[1..6] of integer;  
    x, i, j, k : integer;  
begin  
  StringGrid1.ColCount:=7; //ustalenie ilosci kolumn  
  StringGrid1.RowCount:=7;  
  StringGrid1.Cells[1,0]:='Ust. pocz.';
```

```
for i:=1 to 6 do
  StringGrid1.Cells[i+1,0]:=IntToStr(i)+' przejście';

liczby[1]:=574;
liczby[2]:=303;
liczby[3]:=34;
liczby[4]:=125;
liczby[5]:=8;
liczby[6]:=23;

for j:=1 to 6 do
  StringGrid1.Cells[1,j]:=IntToStr(liczby[j]);

for i:=2 to 6 do
begin
  for j:=6 downto i do
  begin
    if liczby[j-1]>liczby[j] then
    begin
      x := liczby[j-1];
      liczby[j-1]:= liczby[j];
      liczby[j] := x;
    end;

    for k:=i to 6 do
    begin
      StringGrid1.Cells[k,j-1]:=IntToStr(liczby[j-1]);
      StringGrid1.Cells[k,j]:=IntToStr(liczby[j]);
    end;

  end;
end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  close;
end;
end.
```

C++ *Builder* (**Babelki**) – formularz jak na Rys. 8.1.

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  

```

```
liczby[5]=23;

for (j=0; j<=5; j++)
{
  StringGrid1->Cells[1][j
+1]=IntToStr(liczby[j]); //ust.poczatkowe
}

for (i=1; i<=5; i++)
{
  for(j=5; j>=i; --j)
  {
    if (liczby[j-1] > liczby[j])
    {
      x = liczby[j-1];
      liczby[j-1]=liczby[j];
      liczby[j]= x;
    }

    for (k=i; k<=5; k++)
    {
      {StringGrid1->Cells[k+1][j]=IntToStr(liczby[j-1]);
      StringGrid1->Cells[k+1][j+1]=IntToStr(liczby[j]);
    }

    } //k
  } //j
} //i
}
//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Close();
}
//-----
```

Tablice przechowują również teksty

Tablice przechowują nie tylko liczby, ale również teksty (ang. *strings*). Teksty są bardzo użyteczne w programach, gdyż mogą przechowywać informacje takie jak nazwy plików, tytuły książek, nazwiska pracowników i inne ciągi określonych znaków.

Delphi

W języku Turbo Pascal typ łańcuchowy (tekstowy) był ciągiem znaków (tekstem) poprzedzonym liczbą, która wskazywała rozmiar tego łańcucha. Ponieważ długość łańcucha mieściła się w jednym bajcie, to łańcuch mógł mieć maksymalnie tylko 255 znaków. 32-bitowa wersja Delphi ma dodatkowo wbudowany typ długich łańcuchów tekstowych.

W Delphi istnieją trzy typy łańcuchów tekstowych:

1. typ **ShortString**, który odpowiada dawnym typom łańcuchowym w języku Turbo Pascal,
2. typ **ANSIString** odpowiada nowym, o zmiennej długości łańcuchom tekstowym z dynamicznym przydziałem pamięci – rozmiar tych łańcuchów jest praktycznie nieograniczony,
3. typ **WideString** jest podobny do typu ANSIString, również z dynamicznym przydziałem pamięci, lecz jest oparty na znakach typu **WideChar**, który obsługuje znaki wielobajtowe – system *Unicode*.

Z łańcuchami związany jest szereg praktycznych funkcji (dostarczonych przez producentów oprogramowania) pozwalających na swobodne operacje na tekstach. Zilustrujemy to odpowiednimi przykładami, zachęcając Czytelnika do samodzielnego eksperymentowania z pozostałymi funkcjami.

Pierwszy z programów - **Długość łańcucha** - umożliwia odczytywanie długości łańcucha zapamiętanego w odpowiedniej zmiennej.

Delphi (**Długość Łańcucha**)

Formularz

Na formularzu wykonujemy następujące czynności:

1. umieszczamy dwa przyciski **Button**, jeden u góry (**Button1**) i jeden na dole z prawej strony (**Button2**),
2. następnie klikamy myszą raz na **Button1** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button1** na **Zacznij**,
3. klikamy myszą raz na **Button2** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button2** na **Zakończ**,
4. u góry formularza z jego lewej strony umieszczamy komponent **Edit**, w *Object Inspectorze* usuwamy słowo **Edit1** we właściwościach (*Properties*) w *Text*.
5. na dole formularza z jego lewej strony umieszczamy komponent **ListBox**,
6. dwukrotnie klikamy na formularzu przycisk **Zakończ** i podpinamy związaną z nim procedurę **procedure TForm1.Button2Click(Sender: TObject);**

begin

Close;

end;

7. następnie dwukrotnie klikamy na formularzu przycisk **Zaczynij** i podpinamy związaną z nim procedurę **procedure TForm1.Button1Click(Sender: TObject)**, która została opisana w programie poniżej.

Wydruk całego programu.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, StdCtrls;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    Edit1: TEdit;  
    Listbox1: TListBox;  
    Button2: TButton;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;
```

implementation**{ \$R *.dfm }****procedure TForm1.Button1Click(Sender: TObject);****var****imie : AnsiString;****nazwisko : AnsiString;****begin****Edit1.Text:='Program mierzy dlugosc lancucha imie i nazwisko';****imie:=InputBox('','Podaj imie,');****nazwisko:=InputBox('','Podaj nazwisko,');****ListBox1.Items.Clear;****ListBox1.Items.Add('Imie '+imie+' zawiera '+IntToStr(Length(imie))+ ' liter(y).');****ListBox1.Items.Add('Nazwisko '+nazwisko+' zawiera '+IntToStr(Length(nazwisko))+ ' liter(y).');****end;****procedure TForm1.Button2Click(Sender: TObject);****begin****Close;****end;****end.**

W programie powyżej skorzystaliśmy z gotowej funkcji **Length(zmienna_łańcuchowa)**. Wynikiem jest długość łańcucha znaków **zmienna_łańcuchowa**, czyli liczba przedstawiająca liczbę znaków, z których się on składa. Wynik jest typu integer.

C++ Builder

Język C++ Builder dostarcza nowego mechanizmu obsługi łańcuchów znaków – klasę **AnsiString**. Klasa ta posiada szereg metod, które umożliwiają obsługę łańcuchów znakowych (zobacz: Dorobek, 2002). Oto przykład zadeklarowania dwóch zmiennych klasy **AnsiString**:

AnsiString imie, nazwisko;

Program (**Długość łańcucha**) umożliwia odczytywanie długości łańcucha zapamiętanego w odpowiedniej zmiennej.

Formularz

Na formularzu wykonujemy następujące czynności:

1. umieszczamy dwa przyciski **Button**, jeden u góry (**Button1**) i jeden na dole z prawej strony (**Button2**),
2. następnie klikamy myszą raz na **Button1** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button1** na **Zaczynij**,
3. klikamy myszą raz na **Button2** i w *Object Inspectorze* zmieniamy we właściwościach (*Properties*) w *Caption* tekst **Button2** na **Zakończ**,
4. u góry formularza z jego lewej strony umieszczamy komponent **Edit**, w *Object Inspectorze* usuwamy słowo **Edit1** we właściwościach (*Properties*) w *Text*.
5. na dole formularza z jego lewej strony umieszczamy komponent **ListBox**,
6. dwukrotnie klikamy na formularzu przycisk **Zakończ** i podpinamy związaną z nim funkcję **void __fastcall**

```
TForm1::Button2Click(TObject *Sender)
```

```
{
```

```
    Close();
```

```
}
```

7. następnie dwukrotnie klikamy na formularzu przycisk **Zaczynij** i podpinamy związaną z nim funkcję **void __fastcall TForm1::Button1Click(TObject *Sender)**, która została opisana w programie poniżej.

Wydruk całego programu.

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  

```

```
Ansistring imie, nazwisko;

Edit1->Text="Program mierzy dlugosc lancucha imie i
nazwisko";
imie=InputBox("", "Podaj imie", "");
nazwisko=InputBox("", "Podaj nazwisko", "");

ListBox1->Items->Clear();
ListBox1->Items->Add("Imie "+imie+ " zawiera
"+IntToStr(imie.Length())+" liter(y).");
ListBox1->Items->Add("Nazwisko "+nazwisko+ " zawiera
"+IntToStr(nazwisko.Length())+" liter(y).");
}
//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Close();
}
//-----
```

W programie skorzystaliśmy z metody **Length()** klasy **Ansistring**. W następnym przykładzie wykorzystamy możliwości składania łańcuchów.

Łączenie łańcuchów (konkatenacja) w języku Delphi umożliwia nam funkcja **Concat(łańcuch_1, łańcuch_2, łańcuch_n)**, która łączy kilka łańcuchów w jeden łańcuch. Musimy pamiętać, że składanie różnych łańcuchów nie jest przemienne, co łatwo możemy sprawdzić wprowadzając łańcuchy **ma** i **ta**.

Delphi (Concat) – formularz jak wyżej.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, StdCtrls;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    Edit1: TEdit;  
    Listbox1: TListBox;  
    Button2: TButton;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{$R *.dfm}  
  
procedure TForm1.Button1Click(Sender: TObject);  
  
var  
  lancuch_1 : AnsiString;  
  lancuch_2 : AnsiString;
```

```
begin
  Edit1.Text:='Program dodaje dwa łańcuchy';
  lancuch_1:=InputBox('','Podaj pierwszy tekst,');
  lancuch_2:=InputBox('','Podaj drugi tekst,');
  ListBox1.Items.Clear;
  ListBox1.Items.Add('Po dodaniu lancuch_1 + lancuch_2 ->
'+Concat(lancuch_1,lancuch_2));
  ListBox1.Items.Add('Po dodaniu lancuch_2 + lancuch_1 ->
'+Concat(lancuch_2,lancuch_1));
  ListBox1.Items.Add('');
  ListBox1.Items.Add('Składanie łańcuchow nie jest
przemienne');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close;
end;

end.
```

C++ *Builder* (**Concat**) – formularz jak wyżej.

```
//-----
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
```

```
{
}
//-----

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString lancuch_1, lancuch_2;

    Edit1->Text="Program dodaje 2 lancuchy";
    lancuch_1=InputBox("", "Podaj pierwszy tekst", "");
    lancuch_2=InputBox("", "Podaj drugi tekst", "");

    ListBox1->Items->Clear();
    ListBox1->Items->Add("Po dodaniu
lancuch_1+lancuch_2->"+(lancuch_1+lancuch_2));
    ListBox1->Items->Add("Po dodaniu
lancuch_2+lancuch_1->"+(lancuch_2+lancuch_1));
    ListBox1->Items->Add("");
    ListBox1->Items->Add("Skladanie lancuchow nie jest
przemienne");
}
//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Close();
}
//-----
```

Na koniec zaproponujemy następujące zadanie: *Mając na wejściu ciąg napisów np. qqqq, asd, asdgfter, wGee, yuiyu, weerw, f, ytu4df należy wyprowadzić elementy o nieparzystej długości oraz długość i nr elementu najdłuższego i najkrótszego³. Rozwiązanie dla obu języków programowania znajduje się poniżej.*

³Odpowiedź: na wyjściu powinniśmy otrzymać następujące wynik asd, yuiyu, weerw, f, 8, 3, 1, 7.

Delphi (Zadanie)

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes,  
  Graphics, Controls, Forms,  
  Dialogs, StdCtrls;  
  
type  
  TForm1 = class(TForm)  
    Button1: TButton;  
    Button2: TButton;  
    Listbox1: TListBox;  
    procedure Button2Click(Sender: TObject);  
    procedure Button1Click(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{$R *.dfm}  
  
procedure TForm1.Button2Click(Sender: TObject);  
begin  
  Close;  
end;  
  
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
tekst: array[1..8] of string;
i, max, min, j, k : integer;
begin
tekst[1]:='qqqq';
tekst[2]:='asd';
tekst[3]:='asdgfter';
tekst[4]:='wGee';
tekst[5]:='yuiyu';
tekst[6]:='weerw';
tekst[7]:='f';
tekst[8]:='ytu4df';

max:=1; min:=1;
ListBox1.Items.Clear();

for i:=1 to 8 do
begin
if (Length(tekst[i]) mod 2 <> 0)then
ListBox1.Items.Add(tekst[i]);
if max < Length(tekst[i]) then
begin
max:=Length(tekst[i]);
j:=i;
end;
if Length(tekst[i]) <= min then
begin
min:=Length(tekst[i]);
k:=i;
end;
end;
ListBox1.Items.Add(IntToStr(max)+' '+IntToStr(j)+'
'+IntToStr(min)+' '+IntToStr(k));

end;

end.
```

C++ Builder (Zadanie)

```
//-----  
  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----  
  
void __fastcall TForm1::Button2Click(TObject *Sender)  
{  
    Close();  
}  
//-----  
  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    AnsiString tekst[8][9];  
    int i, max, min, j, k, l;  
  
    tekst[0][8]="qqqq";  
    tekst[1][8]="asd";  
    tekst[2][8]="asdgfter";  
    tekst[3][8]="wGee";  
    tekst[4][8]="yuiyu";  
    tekst[5][8]="weerw";  
    tekst[6][8]="f";  
    tekst[7][8]="ytu4df";
```

```
max=1; min=1;  
ListBox1->Items->Clear();  
for (i=0; i < 8; i++)  
{  
    for(j=0; j < 9; j++)  
    {  
        if (IntToStr(tekst[i][j].Length()) % 2 != 0) ListBox1-  
>Items->Add(tekst[i][j]);  
  
        if (max < IntToStr(tekst[i][j].Length()))  
        {  
            max = StrToInt(tekst[i][j].Length());  
            k=i;  
        }  
  
        if (IntToStr(tekst[i][j].Length()) == min)  
        {  
            min = StrToInt(tekst[i][j].Length());  
            l=i;  
        }  
    } //j  
  
} //i  
    ListBox1->Items->Add(IntToStr(max)+" "+IntToStr(k  
+1)+" "+IntToStr(min)+" "+IntToStr(l+1));  
}  
//-----
```

Warto zapamiętać

1. Tablica to taka struktura danych, która umożliwia jednej zmiennej przechowywanie wielu wartości tego samego typu.
2. Deklarując tablicę musimy określić typ wartości, jakie ma ona przechowywać, a także liczbę jej elementów.
3. Wszystkie elementy zadeklarowanej tablicy są tego samego typu.
4. Wstawienie elementu do tablicy odbywa się w programie poprzez podanie numeru indeksu w tablicy, pod którym dana wartość ma być umieszczona.
5. Uzyskanie dostępu do wartości przechowywanej w tablicy odbywa się w programie poprzez podanie nazwy tablicy i numeru elementu.
6. Tablice przechowują również teksty.

Dlaczego warto mieć pełną wersję?



Jak szybko poznać elementy programowania w tych dwóch językach? (dla bardziej zaawansowanych) Standardową metodą jest uczenie się programowania "język po języku". A co powiesz na dwa w jednym? Co gdybyś przy okazji zagłębiania się w szczegóły danej instrukcji w Delphi dowiedział się od razu, jak to samo robi się w C++ Builderze? Te dwa środowiska programistyczne dominują teraz w firmach, to w nich powstaje większość spersonalizowanych aplikacji bazodanowych dla małych i średnich przedsiębiorstw. Odkryj coś, co pozwoli Ci w łatwy sposób budować aplikacje wyglądające jak profesjonalne programy pod Windows... Delphi i C++ Builder to

32-bitowe, w pełni obiektowe, zintegrowane środowiska RAD (ang. Rapid Application Development) do szybkiego tworzenia aplikacji w systemie operacyjnym Windows. W publikacji znajdziesz m. in.: 1. Tablice Dowiesz się, w jaki sposób deklarujemy tablice jedno- i dwuwymiarowe oraz o tym, że tablice przechowują nie tylko liczby, ale również teksty. 2. Podprogramy Dowiesz się, co to są podprogramy i do czego można je wykorzystać. 3. Elementy grafiki Nauczysz się pisać proste programy graficzne w Delphi i C++ Builder. 4. Przechowywanie informacji w rekordach i w strukturach Poznasz informacje o rekordach i strukturach oraz dowiesz się, w jaki sposób te informacje są przechowywane w języku Delphi i C++ Builder. 5. Elementy programowania obiektowego Poznasz między innymi elementy programowania obiektowego. Nauczysz się pisać proste programy zawierające obiekty. 5. Operacje wejścia/wyjścia Dowiesz się, co to są pliki oraz jak zapisywać informacje do pliku i jak je odczytywać.

Pełną wersję książki zamówisz na stronie wydawnictwa Złote Myśli

<http://www.zlotemysli.pl/prod/6312/programuje-w-delphi-i-c-builder-cz-2-miroslaw-j-kubiak.html>