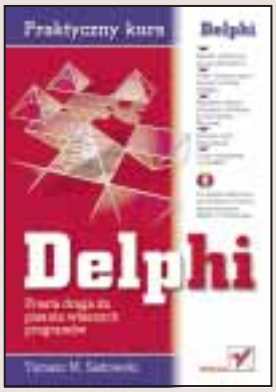
Praktyczny kurs Delphi



Autor: Tomasz M. Sadowski
ISBN: 83-7361-013-8
Format: B5, stron: 396

Tomasz Sadowski, autor wielokrotnie wznawianego, podręcznika „Praktyczny kurs Turbo Pascala", zaprasza Cię tym razem do krainy Delphi. Delphi to system szybkiego tworzenia aplikacji (RAD), w którym programy w dużym stopniu buduje się z gotowych komponentów. O ile Turbo Pascal używany jest głównie w dydaktyce informatyki, o tyle Delphi jest stale rozwijanym i szeroko rozpowszechnionym środowiskiem programistycznym, łączącym przejrzystość ObjectPascala z przyjaznym interfejsem Windows. Być może niektóre z programów używanych przez Ciebie na co dzień zostały napisane właśnie w Delphi.

Książka „Praktyczny kurs Delphi" napisana jest żywym, przystępnym językiem. Położono w niej nacisk raczej na praktykę niż teorię, i dlatego to wspaniałe wprowadzenie w programowanie dla osób, które wcześniej nie zetknęły się z tą tematyką. Już po przeczytaniu kilku stron będziesz mógł napisać pierwszy program!

Dzięki książce poznasz:

- Środowisko Delphi
- Język ObjectPascal: jego podstawowe konstrukcje i typy danych
- Komponenty zawarte w Delphi
- Sposoby pisania własnych funkcji i procedur
- Programowanie obiektowe
- Programowanie grafiki

Po lekturze „Praktycznego kursu Delphi" przekonasz się, że pisanie profesjonalnych aplikacji nie jest tak trudne, jak by się mogło wydawać. Przekonasz się również, że książki poświęcone programowaniu nie muszą być nudną i przepełnioną niezrozumiałą techniczną terminologią lekturą. Ta książka nauczy Cię programować w Delphi – możesz być tego pewien.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 12.

Zapisujemy więcej danych, czyli tablice

- Powtórka ze statystyki
- Typy strukturalne
- Tworzenie i wykorzystanie tablic
- Wektory i macierze

Jak powszechnie wiadomo, komputer jest narzędziem służącym do szybkiego przetwarzania dużych ilości danych. Przedstawione do tej pory programy nie dawały naszemu wielkiemu kalkulatorowi specjalnego pola do popisu — dane, na których operowaliśmy, sprowadzały się zwykle do kilku liczb. Nikogo jednak nie trzeba przekonywać, że większość zadań, z którymi mierzą się komputery w praktyce, wymaga wprowadzania, przechowywania i przetwarzania ogromnych ilości danych. Cel, który postawimy sobie na początku tego rozdziału, będzie znacznie mniej ambitny — spróbujemy mianowicie rozwiązać prosty problem statystyczny, jakim jest obliczenie wartości średniej i miary rozrzutu w zadanej grupie pomiarów pewnej wielkości. Obliczanie średniej i odchylenia standardowego (ono właśnie jest miarą rozrzutu) jest również typowym zadaniem inżynierskim, toteż jego zaprogramowanie może przydać się nam w przyszłości.

Jak powszechnie wiadomo, wartość średnia dla N liczb wyraża się wzorem:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

gdzie i jest numerem kolejnej wartości w zestawie. Przełożenie tego zapisu na Pascal jest na pierwszy rzut oka banalne, ale… gdzie właściwie będą przechowywane kolejne

wartości x_i? Wykorzystanie do tego celu pojedynczych zmiennych typu prostego (a tylko takie na razie znamy) jest praktycznie niemożliwe — nawet gdybyśmy uparli się przy zadeklarowaniu zmiennych `x1`, `x2`, `x3`..., nie wiemy *ile* należałoby ich zadeklarować, nie mówiąc już o problemach z odwoływaniem się do nich w pętli sumującej. Zauważmy jednak, że nasze liczby tworzą jednolity ciąg danych tego samego typu, w matematyce zapisywany w postaci tzw. *wektora*:

$$[x_1, x_2, x_3, \ldots, x_N]$$

Być może kompilator potrafi posłużyć się podobną reprezentacją i zna jakiś sposób na „poszufladkowanie" danych w pamięci tak, by dało się odwoływać do nich poprzez kolejny numer w ciągu?

Opisane powyżej rozwiązanie znane jest w programowaniu od kilkudziesięciu lat pod nazwą *tablicy* (ang. *array*). Jest ona zestawem danych tego samego typu, zajmującym pewien (na ogół ciągły) obszar w pamięci i pozwalającym na odwoływanie się do poszczególnych elementów poprzez podanie ich numeru, czyli tzw. *indeksu*. Mówiąc ogólniej, tablica jest *strukturą* złożoną z danych innego typu, dlatego też należy ona do grupy tzw. *strukturalnych typów danych*. Zmienna typu strukturalnego składa się z innych danych, zorganizowanych zgodnie z regułami obowiązującymi dla danego typu (w przypadku tablic jednowymiarowych dane ustawiane są po prostu w pamięci jedna za drugą). Poszczególne elementy zmiennej strukturalnej mogą być typu prostego (np. liczby całkowite) lub strukturalnego (np. tablice lub rekordy) — innymi słowy, możemy tworzyć „piętrowe" dane strukturalne. Dwa najważniejsze typy danych strukturalnych, którymi zajmiemy się w naszej książce, to tablice (grupujące elementy tego samego typu) oraz rekordy (grupujące elementy różnych typów).

Jak każda inna zmienna, tablica identyfikowana jest w programie poprzez nazwę (np. `liczby`), natomiast dostęp do jej poszczególnych elementów odbywa się poprzez podanie ich indeksu. Ten ostatni jest po prostu numerem danego elementu tablicy, zapisanym w nawiasach kwadratowych, np.:

```
x := liczby[4]; { odczytaj czwarty element tablicy liczby }
```

Graficznie można to przedstawić następująco:

indeks	1	2	3	4	5	6
wartość	1,43	1,66	2,01	1,23	1,98	3,11

Jak można się domyślać, przed użyciem tablicę należy zadeklarować, do czego służy słowo kluczowe **`array`**:

```
var
  liczby : array[1..100] of real;
```

Po słowie **`array`** następuje para nawiasów kwadratowych, w których zapisujemy indeks pierwszego i ostatniego elementu tablicy, rozdzielone dwiema kropkami (nie dwukropkiem!). Niezbędne jest również poinformowanie kompilatora o typie poszczególnych elementów tablicy — w naszym przypadku są to wartości typu `real`. Cały powyższy

zapis można przetłumaczyć na polski jako „tablica o stu elementach, numerowanych od 1 do 100, złożona z liczb rzeczywistych". Podobnie jak w przypadku zmiennych prostych, bezpośrednio po zadeklarowaniu elementy tablicy mają wartość przypadkową[1].

Deklarując tablicę należy pamiętać o dwóch ograniczeniach:

- indeksy muszą być stałymi (innymi słowy, w Pascalu nie można deklarować tzw. tablic dynamicznych w sensie znanym np. z BASIC-a),
- indeksy muszą być typu porządkowego (chociaż nie muszą być liczbami — można np. zadeklarować tablicę indeksowaną wartością typu `char`).

W większości zastosowań tablice indeksowane są od jedynki, chociaż Pascal tego nie wymaga — równie dobrze można indeksować tablicę od zera (jak w języku C) lub tysiąca. Rzecz jasna, indeks końcowy musi być większy lub równy początkowemu.

Niestandardowe indeksowanie tablic może być źródłem problemów, bowiem odwołanie do elementu spoza zadeklarowanego przedziału kończy się często trudnymi do wykrycia błędami wykonania. O ile nie ma naprawdę ważnego powodu, najlepiej zawsze stosować jednolity sposób indeksowania (zwykle od jedynki, chociaż programiści piszący w języku C preferują indeksowanie od zera).

Warto przy okazji wspomnieć o funkcjach `Low` i `High`, które dla tablic zwracają odpowiednio wartość pierwszego i ostatniego indeksu. Pozwalają one pominąć przekazywanie informacji o zakresie indeksów, o ile oczywiście mamy pewność, że obliczenia będą zawsze dotyczyły całej tablicy.

Poszczególne elementy tablicy mogą być dowolnego typu (ale wszystkie tego samego), przy czym niekoniecznie musi to być typ prosty, jak `real` czy `boolean` — równie dobrze można zadeklarować tablicę złożoną z elementów typu strukturalnego, np. innych tablic lub rekordów. Przykładowa deklaracja:

```
var
  Macierz : array[1..50] of array [1..50] of real;
```

utworzy dwuwymiarową tablicę złożoną z 50 wektorów (tablic jednowymiarowych), z których każdy zawiera 50 liczb rzeczywistych. Składnia Pascala pozwala na skrócenie powyższego zapisu do postaci:

```
var
  Macierz : array[1..50, 1..50] of real;
```

która lepiej odzwierciedla strukturę tablicy dwuwymiarowej, a przy okazji jest prostsza w zapisie. Możliwe jest także tworzenie tablic trój- i więcejwymiarowych (Pascal nie narzuca ograniczeń na liczbę wymiarów tablicy ani wartości indeksów), jednak stosowane są one rzadko.

[1] Ściślej rzecz biorąc, tablice deklarowane globalnie inicjalizowane są zerami, jednak lepiej wyrobić sobie nawyk dmuchania na zimne i nie zakładać, że świeżo zadeklarowana tablica zawiera sensowne wartości.

Zanim zabierzemy się do pracy, jeszcze jedna uwaga praktyczna. W porównaniu do starych aplikacji 16-bitowych, Delphi umożliwia tworzenie znacznie większych struktur danych (maksymalny rozmiar tablicy w Turbo Pascalu wynosił około 64 kB, w Delphi ograniczony jest w zasadzie wielkością dostępnej pamięci). Deklarując tablice wewnątrz procedur należy jednak pamiętać, że są one tworzone na stosie, którego pojemność jest ograniczona (domyślnie do 1 megabajta). O ile Pascal nie narzuca formalnych ograniczeń na liczbę wymiarów i elementów tablicy, musimy pamiętać o ograniczeniu praktycznym, wynikającym z rozmiarów dostępnej pamięci. Dodatkowym haczykiem jest fakt, iż wszystkie zmienne lokalne tworzone są dopiero w momencie wywołania funkcji, a zatem do przepełnienia stosu (i krytycznego błędu wykonania) dochodzi w dość nieprzewidzianym momencie podczas pracy programu. W przeciwieństwie do lokalnych, tablice deklarowane globalnie mają do dyspozycji całą pamięć dostępną dla programu (znacznie więcej, niż zajmuje stos), zaś zadeklarowanie zbyt dużej struktury powoduje błąd bezpośrednio po uruchomieniu. Innymi słowy, deklaracja:

```
var
  liczby : array[1..1000000] of real; { milion liczb rzeczywistych }
```

ma szansę „przejść" w przypadku umieszczenia jej w sekcji globalnej programu lub modułu, natomiast raczej na pewno spowoduje błąd *wykonania* (nie kompilacji!), jeśli umieścimy ją wewnątrz procedury. Nie znaczy to oczywiście, że tablice najlepiej deklarować jako globalne — pamiętajmy, że zmienna globalna istnieje (i zajmuje pamięć) przez cały czas działania programu, natomiast zmienna lokalna znika po wykonaniu swojego zadania, zwalniając zajętą przez siebie pamięć.

Pora na czyny. Nasz program powinien umożliwić wprowadzenie ciągu liczb o dowolnej długości, zapamiętać go, a następnie wyliczyć i wyświetlić wartość średnią i odchylenie standardowe, dane wzorem:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}$$

Bardziej spostrzegawczy Czytelnicy być może zauważyli, że wyliczenie średniej wcale nie wymaga tablicowania danych, bowiem sumowanie można prowadzić na bieżąco, w miarę wprowadzania kolejnych liczb. Niestety obliczenie odchylenia standardowego wymaga z jednej strony wcześniejszego ustalenia wartości średniej, z drugiej zaś — ponownego dostępu do poszczególnych liczb. Innymi słowy, wprowadzone wartości trzeba gdzieś zapamiętać, przynajmniej na chwilę[2].

[2] Ściślej rzecz biorąc, wcale nie trzeba — kilka przekształceń pozwala zapisać ostatni wzór w postaci

$$\sqrt{\frac{\sum_{i=1}^{N} x_i^2 - \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N}}{N-1}}$$

która, jak widać, nie zawiera odwołań do poszczególnych liczb, a jedynie do ich sumy i sumy kwadratów. Metoda ta wykorzystywana jest powszechnie np. w kalkulatorach, jednak jej zastosowanie odebrałoby nam pretekst do wprowadzenia pojęcia tablicy…

Jako podstawę do budowy naszego programu możemy wykorzystać projekt *Sumowanie* z rozdziału 9. Po dokonaniu kilku modyfikacji (zmiana nazw komponentów wyjściowych i etykiet) powinien on prezentować się następująco:

Co prawda potrafimy już korzystać z procedur i funkcji (a obliczenia statystyczne aż proszą się o ich zastosowanie), ale na razie pójdziemy po linii najmniejszego oporu i całość kodu upakujemy w procedurze obsługi przycisku *Oblicz*:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : integer;                          { numer wiersza zawierającego liczbę }
  liczby : array[1..1000] of real;      { i jej wartość }
  ileLiczb : integer;                   { ile jest liczb - pomijamy puste wiersze }
  suma, sumaKwadr : real;               { obliczane sumy }
  srednia, odchStd : real;              { i statystyki }
begin
  suma := 0.0;                          { zainicjalizuj sumy - bardzo ważne! }
  sumaKwadr := 0.0;
  ileLiczb := 0;                        { zakładamy, że nie podano żadnej liczby }
  { przenieś liczby z pola wejściowego do tablicy liczby }
  for i := 0 to WejLiczby.Lines.Count-1 do     { przejdź po wszystkich wierszach }
    if WejLiczby.Lines[i] <> '' then           { jeśli wiersz zawiera wartość... }
      begin
        Inc(ileLiczb);
        liczby[ileLiczb] := StrToFloat(WejLiczby.Lines[i]); { pobierz liczbę }
      end;
  { oblicz średnią }
  for i := 1 to ileLiczb do
    suma := suma + liczby[i];                          { sumuj liczby }
  srednia := suma/ileLiczb;                            { oblicz statystykę }
  { oblicz odchylenie standardowe }
  for i := 1 to ileLiczb do
    sumaKwadr := sumaKwadr + sqr(liczby[i] - srednia); { sumuj kwadraty odchyleń }
  odchStd := sqrt(sumaKwadr/(ileLiczb-1));             { oblicz statystykę }
  WyjSuma.Text := FloatToStrF(suma, ffFixed, 4, 10);   { wyprowadź sumy }
  WyjSrednia.Text := FloatToStrF(srednia, ffFixed, 4, 10);
  WyjOdchStd.Text := FloatToStrF(odchStd, ffFixed, 4, 10);
end;
```

Same obliczenia nie wymagają chyba komentarza. Deklaracja tablicy `liczby` umożliwia obsłużenie do tysiąca wartości, co na potrzeby tego eksperymentu chyba wystarczy. Warto zauważyć, że na siłę moglibyśmy się obejść bez tablicy, bowiem nasze liczby cały czas przechowywane są w komponencie `WejLiczby` (skąd *notabene* pobieramy je za pomocą operatora indeksowania, tak samo jak z tablic). Nikogo jednak nie

trzeba przekonywać, że pole tekstowe nie jest najlepszym narzędziem do przechowywania liczb rzeczywistych, a poza tym narzut związany z pobieraniem poszczególnych wierszy i przekształcaniem ich na wartości typu `real` jest zbyt duży, by rozwiązanie takie miało sens. Na koniec zwróćmy uwagę na intensywne wykorzystanie w programie pętli **for** — obsługa tablic i dostępu indeksowanego to jej ulubione zastosowania.

Jak widać, wykorzystanie tablic jednowymiarowych (a takich w programach spotyka się najwięcej) nie jest specjalnie skomplikowane. Przejdźmy zatem na kolejny poziom i powiedzmy kilka słów na temat tablic dwuwymiarowych. Wspomniana już deklaracja:

```
var
  Macierz : array[1..50, 1..50] of real;
```

tworzy dwuwymiarową tablicę złożoną o wymiarach 50 na 50 elementów, zawierającą liczby rzeczywiste. W matematyce tablica taka nosi nazwę *macierzy* i może być zapisana symbolicznie jako:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,50} \\ x_{2,1} & x_{2,2} & & x_{2,50} \\ \vdots & \vdots & \vdots & \vdots \\ x_{50,1} & x_{50,2} & \cdots & x_{50,50} \end{bmatrix}$$

Zarówno wektory (tablice jednowymiarowe), jak i macierze stosowane są szeroko w obliczeniach naukowych i inżynierskich, toteż umiejętność posługiwania się nimi może okazać się bardzo istotna. Skoro tak, przedstawimy obecnie kilka podstawowych technik obsługi tablic dwuwymiarowych.

Dostęp do pojedynczego elementu tablicy `Macierz` można zapisać następująco:

```
x := Macierz[4, 6];
```

W powyższym zapisie posługujemy się dwoma indeksami — wiersza i kolumny — rozdzielonymi przecinkiem. W ogólnym przypadku tablicy *N*-wymiarowej nasz zapis miałby postać:

```
x := Tablica[indeks1, indeks2, ... , indeksN];
```

Odwołania do pojedynczych elementów macierzy spotykane są dość rzadko; na ogół obliczenia dotyczą całych wierszy lub kolumn, a indeksy przybierają kolejno wartości od pierwszego do ostatniego elementu wiersza lub kolumny. Ponieważ do indeksowania w pojedynczym wymiarze najlepiej nadaje się pętla **for**, a wymiary są dwa, musimy wykorzystać dwie pętle. Przykładowa konstrukcja tworząca tzw. *macierz jednostkową* (zawiera ona same zera za wyjątkiem głównej przekątnej[3], na której znajdują się jedynki) ma postać:

```
for i := 1 to 50 do
  for j := 1 to 50 do
    if i = j then Macierz[i, j] := 1.0
    else          Macierz[i, j] := 0.0;
```

[3] Przekątną główną macierzy kwadratowej tworzą elementy, dla których indeks wiersza jest równy indeksowi kolumny. Jak łatwo się przekonać, są one ułożone wzdłuż przekątnej biegnącej od lewego górnego do prawego dolnego „wierzchołka" macierzy.

Jak widać, w pętli zewnętrznej, indeksującej wiersze (licznik `i`) znajduje się pętla wewnętrzna, indeksująca kolumny (licznik `j`). W niej z kolei umieszczono instrukcję warunkową, wstawiającą do danego elementu wartość 0 lub 1. Konstrukcja taka nosi nazwę *pętli zagnieżdżonych* i jest powszechnie stosowana w przetwarzaniu tablic dwuwymiarowych. Oczywiście obie pętle sterowane są różnymi licznikami, co zresztą jest dość naturalne. Nie dość tego, próba użycia tej samej zmiennej jako licznika obu pętli (co zdarzało się roztargnionym programistom piszącym np. w Turbo Pascalu) jest w Delphi nielegalna — zapis:

```
for i := 1 to 50 do
  for i := 1 to 50 do { ... }
```

zostanie przez kompilator potraktowany jako próba zmodyfikowania wartości licznika zewnętrznej pętli w jej treści, co doprowadzi do błędu kompilacji.

Przy okazji indeksowania warto wspomnieć o jeszcze jednej interesującej kwestii. Na pozór wydawałoby się, że kolejność pętli w powyższym zapisie nie ma znaczenia — innymi słowy obojętne jest, czy licznik pętli zewnętrznej indeksuje wiersze a wewnętrznej kolumny, czy odwrotnie. Okazuje się jednak, że zamiana liczników pętli miejscami:

```
for j := 1 to 50 do
  for i := 1 to 50 do { ... }
```

ma poważny wpływ na czas wykonania programu! Aby to zaobserwować, należy oczywiście powiększyć rozmiary macierzy (być może trzeba będzie zadeklarować ją jako globalną) i użyć precyzyjnego narzędzia do pomiaru czasu (w naszym przykładzie wykorzystaliśmy klasę `TStoper`, którą przedstawimy szczegółowo w części książki poświęconej programowaniu obiektowemu). Czas wypełnienia tablicy o rozmiarach 1000 na 1000 elementów, zmierzony na komputerze z procesorem AMD Athlon 1.2 GHz, wyniósł 20 milisekund w przypadku indeksowania wzdłuż wierszy i 37 milisekund (czyli prawie dwa razy dłużej!) w przypadku indeksowania wzdłuż kolumn. Dlaczego?

Kluczem do rozwiązania zagadki jest sposób przechowywania zawartości tablicy w pamięci. W Pascalu, podobnie jak w języku C, tablice dwuwymiarowe zapisywane są w pamięci kolejno wierszami[4]. Oznacza to, że następujące po sobie odwołania do kolejnych elementów wiersza będą trafiały pod kolejne adresy pamięci, podczas gdy odwołania „wzdłuż" kolumny będą kierowane pod adresy oddalone od siebie o liczbę bajtów odpowiadającą długości pojedynczego wiersza. Działanie mechanizmów dostępu do pamięci (w tym także stosowane w nowoczesnych komputerach tzw. *pamięci podręczne*) powoduje, że odwołania do adresów leżących „blisko" siebie są znacznie szybsze, niż odwołania do adresów rozproszonych w większym obszarze pamięci. A zatem zapisanie w pętli kolejnych elementów wiersza (położonych jeden za drugim) odbędzie się znacznie szybciej niż zapisanie kolejnych elementów kolumny (oddalonych od siebie o wiele bajtów). Oczywiście na cały proces mają wpływ inne czynniki, jak choćby rozmiar tablicy, jednak ogólna reguła jest zawsze ta sama — indeksując tablicę dwuwymiarową należy robić to odpowiednio do sposobu jej przechowywania w pamięci.

[4] Nie jest to bynajmniej powszechna reguła — w Fortranie tablice zapisywane są kolumnami.

Wróćmy teraz do zagadnień bardziej przyziemnych i zastanówmy się, w jaki sposób obliczyć sumę elementów leżących w „górnym trójkącie" macierzy, tj. na jej przekątnej i ponad nią:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,50} \\ & x_{2,2} & & x_{2,50} \\ \vdots & \vdots & \vdots & \vdots \\ & & \cdots & x_{50,50} \end{bmatrix}$$

Jak łatwo sprawdzić, dla każdego sumowanego elementu spełniony jest warunek:

$$i \leq j$$

gdzie `i` jest indeksem wiersza, zaś `j` — indeksem kolumny. Wykorzystując tę regułę, możemy zapisać sumowanie jako:

```
suma := 0.0;
for i := 1 to 50 do
  for j := 1 to 50 do
    if i <= j then suma := suma + Macierz[i, j];
```

Jak nietrudno wyliczyć, całkowita liczba przebiegów obu pętli wynosi 2500 (50 razy 50), przy czym tylko część z nich (dokładnie 1275) będzie „pełnych" — w pozostałych przypadkach instrukcja warunkowa nie wykona się. Jeśli podejrzewasz, że powyższy zapis da się ulepszyć, masz rację. Zauważmy, że w każdym wierszu sumowaniu podlegają tylko te elementy, dla których numer kolumny (`j`) jest większy lub równy numerowi danego wiersza (`i`). Dlaczego zatem nie zapisać:

```
suma := 0.0;
for i := 1 to 50 do
  for j := i to 50 do
    suma := suma + Macierz[i, j];
```

Nikt przecież nie powiedział, że pętla musi rozpoczynać się od jedynki, ani że do inicjalizacji licznika nie można wykorzystać innej zmiennej (np. licznika pętli zewnętrznej)! W ten sposób obcięliśmy prawie połowę przebiegów, co prawda pustych, ale jednak zajmujących jakiś czas — pomiary wykonane przez autora wykazały przyrost prędkości o ponad 20 procent, co jest zyskiem niebagatelnym.

Opisane tu zabiegi to nic innego, jak *optymalizacja* programu — poprawienie jego wydajności poprzez odpowiednią modyfikację kodu źródłowego. W przypadku sumowania elementów ponad przekątniowych optymalizacja polegała na zmianie struktury pętli, co wyeliminowało część niepotrzebnych obliczeń. Usprawnienie ma w tym przypadku naturę ogólną — jest niezależne od używanego języka programowania, kompilatora czy też sprzętu. Z kolei technika zademonstrowana przy okazji tworzenia macierzy jednostkowej jest specyficzna, wykorzystuje bowiem charakterystyczną dla Pascala organizację danych w pamięci oraz mechanizmy sprzętowe, które w innych komputerach mogą być niedostępne lub działać inaczej[5].

[5] Przemilczmy dyskretnie fakt, iż w zasadzie nie dokonaliśmy tu żadnej optymalizacji — kod był optymalny w swojej pierwotnej wersji, my zaś z premedytacją popsuliśmy go, aby zademonstrować efekt zamiany indeksów.

Zapamiętaj

- Typy złożone z danych innych typów noszą nazwę typów strukturalnych. Należą do nich tablice i rekordy.
- Tablice umożliwiają przechowywanie większych ilości danych tego samego typu.
- Tablice mogą być jedno-lub wielowymiarowe. Tablice jednowymiarowe zwane są wektorami, zaś wielowymiarowe — macierzami.
- Dostęp do elementów tablicy realizuje się poprzez indeksowanie. Indeksy tablicy są wartościami całkowitymi; Pascal nie narzuca ograniczeń na zakres i liczbę indeksów.
- Do indeksowania tablic wykorzystuje się najczęściej pętle `for`.
- Operacje na tablicach często można zoptymalizować, bądź to poprzez zmianę układu kodu źródłowego, bądź też poprzez wykorzystanie specyficznych cech samego komputera.

Pomyśl

1. Napisz procedurę wyszukującą i zwracającą najmniejszą i największą wartość w jednowymiarowej tablicy liczb rzeczywistych.
2. *Transpozycją* macierzy nazywamy zamianę miejscami jej kolumn i wierszy. Napisz procedurę `transp`, dokonującą transpozycji macierzy kwadratowej `M` (macierz kwadratowa ma tyle samo wierszy, ile kolumn).