

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

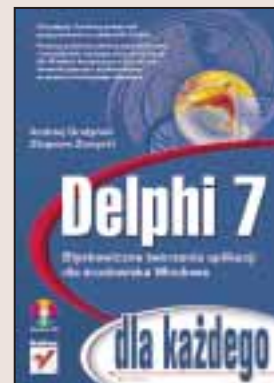
ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# Delphi 7 dla każdego



Autorzy: Andrzej Grażyński, Zbigniew Zarzycki

ISBN: 83-7361-091-X

Format: B5, stron: 830

Zawiera CD-ROM

Dawno już minęły czasy, gdy podstawowym sposobem tworzenia programów było mozolne „wklepywanie” kodu. Forma przekazywanej komputerowi i uzyskiwanej za jego pomocą informacji stała się nie mniej ważna od treści. W takim właśnie kontekście zrodziły się narzędzia do błyskawicznego tworzenia aplikacji (RAD), wśród których jednym z najpopularniejszych jest Delphi. Oparte na języku ObjectPascal ma za sobą długą tradycję, ukazała się już 7 edycja tego narzędzia.

To, że Delphi jest wizualnym środowiskiem programistycznym, w którym wiele działań wykonuje się łatwiej niż w tradycyjnych środowiskach opartych na tekście, nie oznacza, że jego użytkownik może obejść się bez podręcznika. Taki podręcznik trzymasz właśnie w ręku. Został on napisany w sposób fachowy i przystępny. Dzięki „Delphi 7. Dla każdego” poznasz Delphi i nauczysz się pisać w nim programy, nawet jeśli nie jesteś informatykiem.

Książka opisuje:

- Typy danych i operatory w Delphi
- Instrukcje, tworzenie własnych procedur i funkcji
- Programowanie zorientowane obiektowo: klasy, metaklasy, interfejsy
- Tworzenie bibliotek DLL w Delphi
- Środowisko zintegrowane (IDE) Delphi
- Tworzenie atrakcyjnego interfejsu użytkownika
- Śledzenie wykonania programu i usuwanie z niego błędów
- Obsługę baz danych w Delphi

Dzięki narzędziom takim jak Delphi nawet osoby, które wcześniej nie programowały, mogą pisać złożone aplikacje o atrakcyjnym interfejsie. Przy okazji nauczą się podstaw programowania, a zdobytą w ten sposób wiedzę łatwo przeniosą do innych środowisk i systemów.



# Spis treści

|  |           |
|--|-----------|
| <b>Rozdział 1. Wprowadzenie .....</b>                      | <b>9</b>  |
| Trochę zabawy.....   | 11        |
| <b>Rozdział 2. Kompendium języka Delphi.....</b>           | <b>19</b> |
| Moduły aplikacji w Delphi.....                             | 19        |
| Struktury danych w językach programowania.....             | 28        |
| Typy danych w Delphi.....                                  | 29        |
| Typy proste.....   | 30        |
| Typy łańcuchowe.....                                       | 37        |
| Typy strukturalne.....                                     | 42        |
| Typy wariantowe.....                                       | 75        |
| Typy wskaźnikowe.....                                      | 88        |
| Definiowane obszary typów wariantowych.....                | 98        |
| Deklarowanie typów.....                                    | 162       |
| Reprezentacje danych w kodzie źródłowym.....               | 164       |
| Literały.....  | 165       |
| Stałe synonimiczne.....                                    | 166       |
| Stałe typowane.....  | 167       |
| Zmienne.....   | 171       |
| Operatory.....   | 178       |
| Operatory arytmetyczne.....                                | 179       |
| Operatory porównania.....                                  | 181       |
| Operatory logiczne (boolowskie).....                       | 182       |
| Operatory bitowe.....                                      | 182       |
| Operatory zbiorowe.....                                    | 188       |
| Operator konkatencji łańcuchów.....                        | 188       |
| Operator referencji.....                                   | 189       |
| Pierwszeństwo operatorów.....                              | 189       |
| Zgodność typów danych.....                                 | 191       |
| Identyczność typów.....                                    | 191       |
| Kompatybilność typów.....                                  | 192       |
| Zgodność typów w sensie przypisania (przypisywalność)..... | 193       |
| Rzutowanie i konwersja typów.....                          | 195       |
| Instrukcje.....  | 199       |
| Instrukcje proste.....                                     | 200       |
| Instrukcje strukturalne.....                               | 202       |
| Procedury i funkcje.....                                   | 215       |
| Przekazywanie parametrów do procedur i funkcji.....        | 219       |
| Parametry amorficzne.....                                  | 224       |

|  |            |
|--|------------|
| Tablice otwarte.....   | 227        |
| Przeciążanie procedur i funkcji.....                                       | 232        |
| Parametry domyślne procedur i funkcji.....                                 | 236        |
| Zagnieżdżone definicje procedur i funkcji oraz zasięg deklaracji.....      | 239        |
| Deklaracje zapowiadające (forward).....                                    | 240        |
| Typy proceduralne.....   | 241        |
| Obiekty, klasy i programowanie zorientowane obiektowo.....                 | 248        |
| Definiowanie klas.....   | 250        |
| Tworzenie i unicestwianie zmiennych obiektowych.....                       | 260        |
| Zgodność typów obiektowych a polimorfizm.....                              | 263        |
| Przeciążanie metod.....  | 269        |
| Metaklasy.....   | 272        |
| Metaklasy a metody wirtualne.....  | 277        |
| Metaklasy a wirtualne konstruktory.....                                    | 280        |
| Operatory klasowe.....   | 285        |
| Uniwersalne metody metaklasowe.....  | 288        |
| Interfejsy.....  | 289        |
| Deklarowanie interfejsów.....  | 290        |
| Implementowanie interfejsów.....   | 291        |
| Deklaracje zapowiadające klas i interfejsów.....                           | 302        |
| Pocztówka z przeszłości — obiekty Turbo Pascala.....                       | 303        |
| Strukturalna obsługa wyjątków.....   | 304        |
| try...finally, czyli gwarancja.....  | 305        |
| try...except, czyli naprawa.....   | 310        |
| Wyjątki jako klasy Delphi.....   | 317        |
| Hierarchia obsługi wyjątków i wyjątki nieobsłużone.....                    | 324        |
| Generowanie wyjątków.....  | 328        |
| Ponawianie wyjątków.....   | 332        |
| <b>Rozdział 3. Opcje kompilacji i kompilacja warunkowa.....</b>            | <b>337</b> |
| Opcje związane z nowościami Delphi.....                                    | 338        |
| \$H (\$LONGSTRINGS) — długie łańcuchy.....                                 | 338        |
| \$REALCOMPATIBILITY — tradycyjny typ zmiennoprzecinkowy.....               | 339        |
| \$J (\$WRITEABLECONST) — stałe czy zmienne?.....                           | 339        |
| Opcje testowe.....   | 341        |
| \$R (\$RANGECHECKS) — kontrola zakresu.....                                | 342        |
| \$I (\$IOCHECKS) — kontrola poprawności operacji wejścia-wyjścia.....      | 342        |
| \$Q (\$OVERFLOWCHECKS) — kontrola nadmiaru stałoprzecinkowego.....         | 342        |
| \$C (\$ASSERTIONS) — honorowanie albo ignorowanie asercji.....             | 343        |
| Tradycyjne opcje pascalowe.....  | 343        |
| \$X (\$EXTENDEDSTYNTAX) — rozszerzona składnia.....                        | 344        |
| \$V (\$VARSTRINGCHECKS) — kontrola zgodności parametrów łańcuchowych.....  | 344        |
| \$P (\$OPENSTRINGS) — domyślne łańcuchy otwarte.....                       | 346        |
| Opcje interpretacyjne.....   | 346        |
| \$B (\$BOOLEVAL) — obliczanie wyrażeń boolowskich.....                     | 346        |
| \$T (\$TYPEDADDRESS) — kontrola referencji.....                            | 348        |
| Opcje generacyjne.....   | 349        |
| \$O (\$OPTIMIZATION) — optymalizowanie generowanego kodu.....              | 349        |
| \$W (\$STACKFRAMES) — generowanie ramek stosu.....                         | 349        |
| \$A (\$ALIGN) — wyrównywanie pól rekordów i klas.....                      | 350        |
| \$Z (\$MNENUMSIZE) — minimalny rozmiar zmiennej typu wyczerpieniowego..... | 350        |
| \$U (\$SAFEDIVIDE).....  | 351        |
| Opcje sterujące informacją dodatkową.....                                  | 351        |
| \$D (\$DEBUGINFO) — generowanie informacji dla debuggera.....              | 352        |
| \$L (\$LOCALSYMBOLS) — generowanie informacji o symbolach lokalnych.....   | 352        |

|   |            |
|---|------------|
| \$Y (\$REFERENCEINFO i \$DEFINITIONINFO)                                      |            |
| — generowanie informacji o symbolach i odwołaniach do nich.....               | 352        |
| \$M (\$TYPEINFO) — generowanie informacji RTTI.....                           | 353        |
| Opcje związane z komunikatami kompilatora.....                                | 353        |
| \$HINTS.....  | 353        |
| \$WARNINGS.....   | 353        |
| \$WARN.....   | 354        |
| Dyrektywa \$MESSAGE.....  | 355        |
| Opcje parametryczne.....  | 355        |
| \$M (\$MINSTACKSIZE, \$MAXSTACKSIZE)  |            |
| — ustalenie wielkości stosu dla programu.....                                 | 355        |
| \$IMAGEBASE — bazowy obszar ładowania biblioteki DLL.....                     | 356        |
| \$APPTYPE — typ aplikacji.....  | 357        |
| \$D (\$DESCRIPTION) — opis aplikacji.....                                     | 357        |
| \$E (\$EXTENSION) — rozszerzenie generowanego pliku wykonywalnego.....        | 357        |
| Opcje integracyjne.....   | 357        |
| \$I (\$INCLUDE) — dołączanie fragmentów kodu źródłowego.....                  | 357        |
| \$L (\$LINK) — dołączanie skompilowanych modułów.....                         | 363        |
| \$R (\$RESOURCE) — dołączanie plików zasobowych.....                          | 363        |
| Kompilacja warunkowa.....   | 363        |
| Symbole kompilacji warunkowej.....  | 364        |
| Wyrażenia kompilacji warunkowej.....  | 374        |
| Opcje kompilacji i kompilacja warunkowa aktualność modułów wynikowych.....    | 377        |
| <b>Rozdział 4. Biblioteki DLL.....</b>  | <b>379</b> |
| Biblioteki DLL a środowisko zintegrowane Delphi.....                          | 380        |
| Tworzenie bibliotek DLL w Delphi.....   | 382        |
| Klauzule resident, export i local.....  | 389        |
| Statyczne łączenie bibliotek DLL.....   | 393        |
| Moduły importowe bibliotek DLL.....   | 397        |
| Dynamiczne łączenie bibliotek DLL.....  | 403        |
| Procedura inicjująco-kończąca biblioteki DLL.....                             | 406        |
| Bazowy adres ładowania biblioteki DLL.....                                    | 409        |
| Klasy i obiekty w bibliotekach DLL.....                                       | 411        |
| Importowanie obiektu na podstawie deklaracji klasy.....                       | 411        |
| Implementowanie interfejsów przez obiekt znajdujący się w bibliotece DLL..... | 419        |
| <b>Rozdział 5. Środowisko zintegrowane Delphi 7.....</b>                      | <b>429</b> |
| Projekty w środowisku IDE.....  | 430        |
| Domyślne opcje projektu.....  | 436        |
| Opcje menu głównego i paski narzędzi.....                                     | 448        |
| Formularze i paleta komponentów.....  | 449        |
| Podstawowe właściwości i zdarzenia formularzy.....                            | 450        |
| Modalne i niemodalne wyświetlanie formularza.....                             | 461        |
| Ważniejsze metody formularzy.....   | 470        |
| Wielokrotne wykorzystywanie formularzy za pośrednictwem repozytorium.....     | 473        |
| Siatka formularza.....  | 477        |
| Zaznaczanie komponentów.....  | 478        |
| Przesuwanie komponentów.....  | 482        |
| Zmiana rozmiarów komponentów.....   | 483        |
| Skalowanie położenia i rozmiarów komponentów                                  |            |
| za pomocą okna dialogowego skalowania.....                                    | 484        |
| Wyrównywanie i dopasowywanie położenia komponentów.....                       | 485        |
| Właściwości komponentów odpowiedzialne za ich rozmiary i ułożenie.....        | 488        |

|  |            |
|--|------------|
| Ochrona położenia i rozmiarów komponentów .....                              | 491        |
| Wycinanie, kopiowanie i wklejanie komponentów .....                          | 491        |
| Warstwowy układ komponentów .....  | 493        |
| Cykl Tab .....   | 494        |
| Inspektor obiektów .....   | 495        |
| System menu aplikacji i projektant menu .....                                | 498        |
| Zachowywanie układu pulpitu .....  | 518        |
| Edytor kodu Delphi .....   | 520        |
| Otwieranie i zapisywanie plików .....  | 521        |
| Praca z blokami tekstu .....   | 522        |
| Cofanie i ponawianie poleceń ( <i>Undo</i> ) .....                           | 527        |
| Wyszukiwanie i zamiana fragmentów tekstu .....                               | 527        |
| Szablony kodu .....  | 531        |
| Uzupełnianie i parametryzowanie kodu .....                                   | 533        |
| Podpowiedzi kontekstowe związane z elementami kodu .....                     | 535        |
| Uzupełnianie klas .....  | 536        |
| Nawigowanie po implementacji klasy .....                                     | 538        |
| „Parowanie” nawiasów .....   | 538        |
| Menu kontekstowe edytora kodu .....  | 538        |
| Diagramy powiązań .....  | 538        |
| Konfigurowanie edytora kodu .....  | 541        |
| Eksplorator kodu .....   | 549        |
| Ustawienia związane z eksploratorem kodu .....                               | 550        |
| Przeglądarka projektu .....  | 551        |
| <b>Rozdział 6. Śledzenie programu.....</b>                                   | <b>553</b> |
| Przygotowanie aplikacji do śledzenia zintegrowanego .....                    | 554        |
| Elementy śledzenia zintegrowanego .....                                      | 554        |
| Praca krokowa .....  | 554        |
| Punkty przerwań .....  | 557        |
| Podgląd wyrażeń i modyfikowanie zmiennych programu .....                     | 563        |
| Inspektor śledzenia .....  | 569        |
| Śledzenie kodu biblioteki DLL .....  | 571        |
| Dziennik zdarzeń .....   | 572        |
| Ustawienia związane ze zintegrowanym debuggerem .....                        | 573        |
| Strona General .....   | 573        |
| Strona Event Log .....   | 574        |
| Strona Language Exceptions .....   | 574        |
| Strona OS Exceptions .....   | 575        |
| Turbo Debugger — TD32.EXE .....  | 576        |
| <b>Rozdział 7. Komponenty Delphi.....</b>                                    | <b>579</b> |
| Zdarzenia komponentów .....  | 581        |
| Hierarchia komponentów .....   | 587        |
| Komponenty wizualne .....  | 589        |
| Właściwości komponentów oraz ich obsługa za pomocą inspektora obiektów ..... | 594        |
| Właściwości proste .....   | 594        |
| Właściwości wyliczeniowe .....   | 594        |
| Właściwości zbiorowe .....   | 594        |
| Właściwości obiektowe .....  | 595        |
| Właściwości tablicowe .....  | 596        |
| Strumieniowanie komponentów i domyślna wartość właściwości .....             | 598        |
| Współdzielenie metod dostępowych — właściwości indeksowane .....             | 603        |

|   |            |
|---|------------|
| Przegląd komponentów .....  | 605        |
| Etykiety — TLabel i TStaticText .....                             | 606        |
| Komponenty edycyjne — TEdit, TMaskEdit, TMemo i TRichEdit .....   | 607        |
| Przyciski .....   | 616        |
| Komponenty selekcyjne — TListBox, TListCheckBox i TComboBox ..... | 627        |
| Komponent zegarowy — TTimer .....                                 | 631        |
| Komponenty standardowych okien dialogowych .....                  | 632        |
| Tworzenie nowego komponentu .....                                 | 644        |
| <b>Rozdział 8. Technologia COM .....</b>                          | <b>655</b> |
| <b>Rozdział 9. Obsługa baz danych w Delphi .....</b>              | <b>667</b> |
| Wstęp .....   | 667        |
| Lokalne bazy danych .....   | 669        |
| Bazy danych typu klient-serwer .....                              | 669        |
| Wielowarstwowa architektura baz danych .....                      | 670        |
| Przegląd technologii .....  | 671        |
| ClientDataSet .....   | 672        |
| Borland Database Engine (BDE) .....                               | 672        |
| InterBase Express .....   | 673        |
| dbExpress .....   | 674        |
| DbGo (ADOExpress) .....   | 674        |
| DataSnap .....  | 675        |
| Wybór technologii dostępu do danych .....                         | 675        |
| Podejście prototypowe .....                                       | 675        |
| Planowanie „cyklu życiowego” .....                                | 676        |
| Połączenie z bazami danych w środowisku Delphi .....              | 676        |
| Tworzenie prostego formularza bazy danych .....                   | 678        |
| Dodawanie kolejnych kontroltek bazodanowych .....                 | 682        |
| Relacja ogóln-szczególne .....                                    | 685        |
| Obsługa pól rekordów .....  | 687        |
| Właściwości pól i komponent TField .....                          | 687        |
| Edytor właściwości pól .....                                      | 689        |
| Modyfikowanie właściwości pola .....                              | 691        |
| Formatowanie pól przy użyciu masek edycyjnych .....               | 692        |
| Dostęp do wartości kolumny .....                                  | 694        |
| Pola wyliczane .....  | 696        |
| Pola przeglądowe .....  | 698        |
| Weryfikacja danych wejściowych .....                              | 700        |
| Zbiory danych .....   | 702        |
| Kontrolowanie wskaźnika bieżącego rekordu .....                   | 704        |
| Edycja danych .....   | 706        |
| Ograniczanie zbiorów danych .....                                 | 707        |
| Wyszukiwanie rekordów .....                                       | 709        |
| Oznaczanie rekordów za pomocą zakładek .....                      | 711        |
| Definiowanie wartości domyślnych pól .....                        | 712        |
| Podstawowe właściwości, metody i zdarzenia zbiorów danych .....   | 713        |
| Współpraca z serwerami .....                                      | 713        |
| Autoryzacja klienta .....   | 715        |
| Transakcje .....  | 719        |
| Komponent ClientDataSet .....                                     | 721        |
| Borland Database Engine .....                                     | 729        |
| Administrator BDE .....   | 729        |
| Instalacja BDE .....  | 736        |

---

|   |            |
|---|------------|
| Kreator formularzy baz danych .....                       | 736        |
| Komponenty BDE .....                                      | 741        |
| Funkcje BDE API .....                                     | 751        |
| ActiveX Database Objects .....                            | 753        |
| ADO w Delphi .....  | 754        |
| Standardowe sterowniki ADO .....                          | 757        |
| Argumenty połączenia .....                                | 757        |
| TADOCnnction .....  | 759        |
| TADODataset .....   | 761        |
| Excel jako baza danych .....                              | 762        |
| Dostęp do danych za pomocą technologii dbExpress .....    | 766        |
| Komponenty interfejsu dbExpress .....                     | 767        |
| Jak to działa w praktyce? .....                           | 768        |
| Uzgadnianie błędów serwera .....                          | 771        |
| Rozpowszechnianie aplikacji z interfejsem dbExpress ..... | 772        |
| InterBase Express .....                                   | 774        |
| Przegląd komponentów InterBase Express .....              | 775        |
| Technologia DataSnap .....                                | 780        |
| Architektura wielowarstwowa .....                         | 781        |
| MIDAS i DataSnap .....                                    | 783        |
| <b>Podsumowanie .....</b>                                 | <b>791</b> |
| <b>Skorowidz .....</b>                                    | <b>793</b> |

# Typy danych w Delphi

Ile wagonów potrzeba do przewiezienia 14 obrabiarek, jeżeli w jednym wagonie mieszczą się cztery obrabiarki? Wynikająca z prostego dzielenia odpowiedź — 3,5 — jest w oczywisty sposób bezsensowna, nie można bowiem podzielić wagonu na pół (na uparte go można, lecz wtedy nie będzie nadawał się do przewiezienia czegokolwiek).

Jaka jest różnica między liczbami 6,30 a 2,45 — jaki jest odstęp czasowy między godziną 6:30 a 2:45? Argumenty odejmowania niby te same, lecz wyniki różne (3,85 i 3:45).

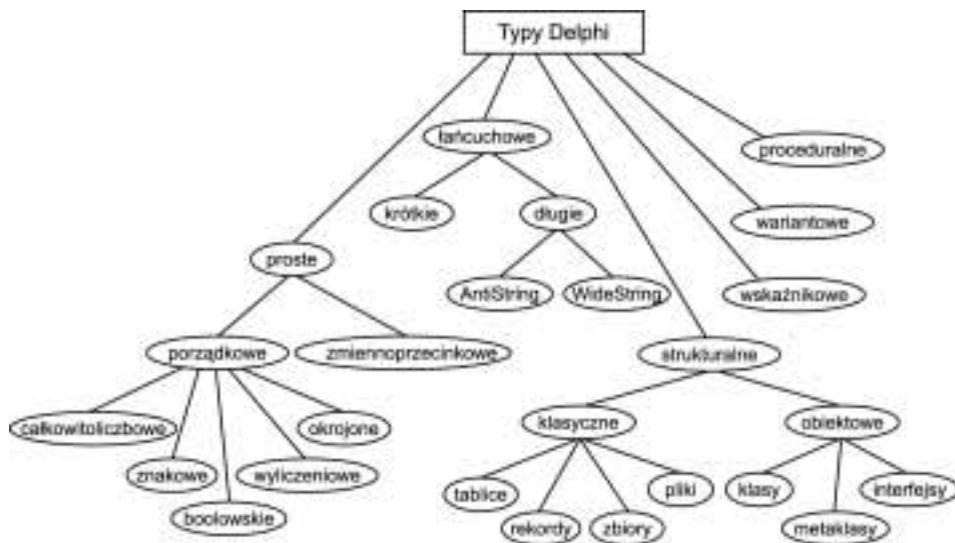
Te banalne przykłady pokazują, iż w wielu obliczeniach musimy ograniczyć się tylko do pewnej kategorii danych (tu liczb całkowitych), a sposób wykonywania podstawowych działań często zależy od charakteru danych (inaczej odejmuje się ułamki dziesiętne, inaczej wskazania zegara).

Gdy przyjrzeć się operacjom wykonywanym przez współczesne procesory, można zauważyć podobną tendencję: zupełnie różne reguły rządzą podstawowymi operacjami matematycznymi wykonywanymi na liczbach całkowitych, liczbach zmiennoprzecinkowych i liczbach dziesiętnych, mimo kodowania ich w tej samej postaci — wzorców bitowych. Ta cecha koncepcyjna (i konstrukcyjna) komputerów zaważyła w głównej mierze na konstrukcji samych języków programowania: okazało się mianowicie, że tworzenie języków programowania będzie łatwiejsze (same zaś języki — bardziej efektywne), jeżeli również na ich gruncie odzwierciedlone zostanie owo zróżnicowanie danych. I tak zostało do dziś, choć gwoli sprawiedliwości wspomnieć należy o dwóch istotnych faktach. Otóż po pierwsze, już w latach sześćdziesiątych twórcy języka Algol 60 zamierzali stworzyć język umożliwiający posługiwanie się liczbami bez konieczności ich różnicowania (choć z możliwością różnicowania na żądanie), lecz w sytuacji, gdy pamięć typowego komputera miała pojemność co najwyżej *kilku tysięcy* słów, tak ambitne zamierzenia musiały pozostać w sferze marzeń. Po drugie natomiast — rozwijające się techniki wymiany danych i oprogramowania pomiędzy komputerami doprowadziły do sytuacji, w której w pewnych zastosowaniach opisanego różnicowania utrzymać się nie da, wymianie podlega bowiem informacja o wysoce zróżnicowanym charakterze. Na potrzeby obsługi tego typu informacji wprowadzono do języków programowania tzw. *zmienne wariantowe*, którymi zajmiemy się w dalszym ciągu rozdziału.

Zbiór wartości o określonym charakterze i zdefiniowanych działaniach nazywamy w języku programowania *typem danych*. Język Delphi ma wiele typów predefiniowanych, z których programista może korzystać bez ich deklarowania. Użytkownik ma jednak nieograniczoną praktycznie możliwość definiowania własnych typów, stosownie do charakteru rzeczywistych danych, odzwierciedlanych (w założeniu możliwie jak najwierniej) w tworzonym programie. Nie jest niczym niezwykłym fakt, iż repertuar tych typów jest nieporównywalnie bogatszy w porównaniu do „gołego” komputera — na tym w końcu polega rola języków wysokiego poziomu.

Hierarchiczne zestawienie typów dostępnych w Delphi przedstawiamy na rysunku 2.1. Ograniczony zakres niniejszej książki nie pozwala na ich wyczerpujące omówienie, skoncentrujemy się więc na ich najważniejszych cechach.





Rysunek 2.1. Zestawienie typów Delphi

## Typy proste

Wielkość typu prostego (zwanego również typem *skalarnym*) zawiera *pojedynczą* wartość. Wielkością skalarną jest więc liczba całkowita, liczba rzeczywista, numer miesiaca czy nawet odpowiedź *tak-nie* na kategoriycznie sformułowane pytanie; nie należą do wielkości skalarnych liczby zespolone (dwie wartości), wektory, macierze, ciągi itp. Pascal nie zalicza również do typów skalarnych napisów, ponieważ każdy z nich rozpatrywany może być jako ciąg wielu znaków.

Typ skalarny może mieć własność *przeliczalności* — określenie to oznacza, że wszystkie jego wartości można ustawić w ciąg, a dla każdej wartości istnieje ściśle określony następnik i poprzednik (jednak z zastrzeżeniem skończoności — patrz niżej). W naturze własność tę mają np. liczby całkowite. *Nie* są przeliczalne liczby rzeczywiste — przeprowadzenie dowodu tego faktu (tzw. metodą przekątniową Cantora) jest przedmiotem elementarnego kursu arytmetyki i wykracza poza ramy niniejszej książki. Typ skalarny posiadający własność przeliczalności nazywamy typem *porządkowym* (*ordinal type*).

Języki programowania ze zrozumiałych względów stanowią jedynie przybliżony opis świata rzeczywistego. Jednym z aspektów owego *przybliżenia* jest *skończoność* — i tak, na przykład, każdy typ całkowitoliczbowy posiada wartość najmniejszą i największą, a liczby zmiennoprzecinkowe mogą być reprezentowane jedynie ze skończoną dokładnością. Jako że wartości typów zmiennoprzecinkowych reprezentowane są przy wykorzystaniu skończonej liczby bitów, liczba możliwych do zapisania wartości jest skończona i można by pokusić się o sztuczne wprowadzanie przeliczalności tych typów; posunięcie takie byłoby jednak ściśle związane z konkretną architekturą komputera i raczej mało przydatne. Z tego względu typy zmiennoprzecinkowe *nie* są w Pascalu zaliczane do typów porządkowych.

## Typy porządkowe

Każdy typ porządkowy reprezentuje w Pascalu skończony, jednoznacznie uporządkowany<sup>1</sup> zbiór wartości. Dla każdego typu porządkowego można określić wartość najmniejszą i największą, zaś dla każdej jego wartości można określić wartość następną i poprzednią (z oczywistym ograniczeniem w odniesieniu do wartości najmniejszej i największej). W Delphi rozróżnia się pięć rodzajów typów porządkowych:

- ♦ całkowitoliczbowe — reprezentujące określone przedziały liczb całkowitych,
- ♦ znakowe — reprezentujące zestawy znaków z określonych alfabetów,
- ♦ logiczne (boolowskie) — reprezentujące wartości *prawda* i *fałsz* w określonych reprezentacjach,
- ♦ wyliczeniowe — reprezentujące podane *explicite* zbiory wartości definiowane przez użytkownika,
- ♦ okrojone — stanowiące określone przedziały innych typów porządkowych.

Dla każdego typu porządkowego określone są następujące funkcje:

- ♦ `Low` — zwraca najmniejszą wartość reprezentowaną przez dany typ,
- ♦ `High` — zwraca największą wartość reprezentowaną przez dany typ.

Dla każdej *wartości* typu porządkowego dodatkowo określone są następujące funkcje:

- ♦ `Ord` — zwraca numer kolejny wartości w ramach wartości reprezentowanych przez dany typ<sup>2</sup>; najmniejsza reprezentowana wartość ma numer 0,
- ♦ `Pred` — zwraca poprzednią wartość w ramach typu (nie dotyczy najmniejszej wielkości reprezentowanej przez dany typ),
- ♦ `Succ` — zwraca następną wartość w ramach typu (nie dotyczy największej wielkości reprezentowanej przez dany typ).

Dla wygody operowania danymi typów porządkowych zdefiniowano również w Pascalu dwie następujące procedury:

- ♦ `Inc` — zwiększa (*inkrementuje*) wartość argumentu (`Inc(X)` równoważne jest `X := Succ(X)`),
- ♦ `Dec` — zmniejsza (*dekrementuje*) wartość argumentu (`Dec(X)` równoważne jest `X := Pred(X)`).

<sup>1</sup> Słowo *jednoznacznie* jest tu istotne; w przeciwieństwie bowiem do skończonego zbioru, np. ułamków zwykłych, który to zbiór uporządkować można według różnych kryteriów, każdy typ porządkowy posiada jedno i tylko jedno uporządkowanie.

<sup>2</sup> Funkcja `Ord()` nie ma zastosowania do typu `Int64` z prostego powodu: rodzimą arytmetyką Delphi jest arytmetyka 32-bitowa i wartość zwracana przez funkcję `Ord()` musi zmieścić się na 32 bitach. Typ `Int64` — jako reprezentowany na 64 bitach — jest więc pewnym wyjątkiem wśród typów porządkowych.

## Typy całkowitoliczbowe

Typ całkowitoliczbowy reprezentuje (zgodnie z nazwą) przedział liczb całkowitych. Zestaw typów całkowitoliczbowych zdefiniowanych w Delphi przedstawiamy w tabeli 2.1.

**Tabela 2.1.** Typy całkowitoliczbowe Delphi

| Typ      | Zakres                    | Reprezentacja maszynowa        |
|----------|---------------------------|--------------------------------|
| Integer  | -2147483648 .. 2147483647 | słowo (32 bity) ze znakiem     |
| Cardinal | 0 .. 4294967295           | słowo (32 bity) bez znaku      |
| Shortint | -128 .. 127               | bajt (8 bitów) ze znakiem      |
| Byte     | 0 .. 255                  | bajt (8 bitów) bez znaku       |
| Smallint | -32768 .. 32767           | półsłowo (16 bitów) ze znakiem |
| Word     | 0 .. 65535                | półsłowo (16 bitów) bez znaku  |
| Longint  | -2147483648 .. 2147483647 | słowo (32 bity) ze znakiem     |
| Longword | 0 .. 4294967295           | słowo (32 bity) bez znaku      |
| Int64    | -263 .. 263-1             | dwusłowo (64 bity) ze znakiem  |

Zwróć uwagę, że typy 32-bitowe (zarówno ze znakiem, jak i bez) posiadają *podwójną* reprezentację. Wynika to z przyjętej w Delphi filozofii, zgodnie z którą typy Integer i Cardinal zawsze cechują się najbardziej optymalną implementacją dla danej wersji Delphi i z tej racji określane są mianem typów rodzinnych (*generic*); ich reprezentacja zależna jest od konkretnej wersji Delphi. Pozostałe typy całkowitoliczbowe, określane mianem typów *fundamentalnych*, mają reprezentację uniwersalną<sup>3</sup>, która w przyszłych wersjach Delphi pozostanie niezmienną.

Należy zachować pewną ostrożność w operowaniu danymi typu Int64. Przekracza on granice domyślnej dla Delphi, 32-bitowej arytmetyki, co ma swe konsekwencje, między innymi, w stosunku do niektórych funkcji i procedur, obcinających argumenty do 32 bitów. Ograniczenie to nie dotyczy jednak funkcji (procedur) standardowych (podajemy za dokumentacją Delphi 7) High, Low, Succ, Pred, Inc, Dec, IntToStr i IntToHex.

Ponadto w sytuacji, gdy wyniki pośrednie obliczeń na liczbach całkowitych przekraczają granice 32 bitów, końcowe wyniki mogą być niepoprawne. W poniższym przykładzie:

```

i, j: Longint;
k: Int64;

...

i := 2000000000;
j := 2000000000;

k := i + j;

```

<sup>3</sup> Pewnym wyłomem w tej uniwersalności jest typ Cardinal, który w Delphi 3 ograniczony był do nieujemnej połówki typu Longint (0 .. 2147483647). Począwszy od Delphi 4 jest on pełnoprawną 32-bitową liczbą bez znaku.

wartością zmiennej *k* po wykonaniu obliczeń jest nie 4000000000, jak można by oczekiwać, lecz -294967296 — wszystko wskutek obcięcia do 32 bitów. By uniknąć tej niespodzianki, należałoby wymusić (na kompilatorze) użycie arytmetyki 64-bitowej, na przykład przez rzutowanie jednego z argumentów dodawania na typ `Int64`:

```
k := Int64(i) + j;
```

Możesz się o tym przekonać, uruchamiając projekt `Arith64` znajdujący się na dołączonym do książki CD-ROM-ie.

## Typy logiczne (boolowskie)

Wielkość typu boolowskiego reprezentuje jedną z wartości *prawda* albo *fałsz*, oznaczonych stałymi symbolicznymi (odpowiednio) `TRUE` i `FALSE`. W Delphi istnieją cztery typy boolowskie (patrz tabela 2.2):

**Tabela 2.2.** Typy boolowskie boolowskie Delphi

| Typ                   | Wielkość            | Reprezentacja wartości FALSE | Reprezentacja wartości TRUE |
|-----------------------|---------------------|------------------------------|-----------------------------|
| <code>Boolean</code>  | bajt (8 bitów)      | 0                            | 1                           |
| <code>ByteBool</code> | bajt (8 bitów)      | 0                            | dowolna wartość niezerowa   |
| <code>WordBool</code> | półsłowo (16 bitów) | 0                            | dowolna wartość niezerowa   |
| <code>LongBool</code> | słowo (32 bity)     | 0                            | dowolna wartość niezerowa   |

Typ `Boolean` jest rodzimym typem pascalowym i jako taki zalecany jest do wykorzystywania w tworzonych aplikacjach. Pozostałe typy boolowskie istnieją przez zgodność z innymi językami (m.in. Visual Basicem) i bibliotekami oprogramowania.

Zwróć uwagę, iż typ `Boolean` cechuje się pewną *niedookreślonością*: reprezentacją `FALSE` jest 0, reprezentacją `TRUE` jest 1. A co z pozostałymi wartościami, od 2 do 255?

Jeżeli *X* jest zmienną typu `Boolean`, warunek:

```
if(X) ...
```

jest równoważny warunkowi:

```
if (Ord(X) <> 0)
```

czyli wspomniany przedział zaliczany jest do reprezentacji wartości `TRUE`. Jednakże już porównanie:

```
if (X = TRUE)
```

interpretowane jest dosłownie, czyli równoważne jest porównaniu:

```
if (Ord(X) = 1)
```

a więc wspomniany przedział zaliczany jest na poczet reprezentacji wartości `FALSE`.

Typy `ByteBool`, `WordBool` i `LongBool` wolne są od tej niekonsekwencji — w odniesieniu do nich porównania:

```
if (X = TRUE)
```

oraz:

```
if (X)
```

są równoważne.

Jeżeli chodzi o skrajne wartości typów boolowskich, sprawa przedstawia się dosyć ciekawie (tabela 2.3):

**Tabela 2.3.** Zakres typów boolowskich

| Typ                   | Ord(Low(typ)) | Ord(High(typ)) |
|-----------------------|---------------|----------------|
| <code>Boolean</code>  | 0             | 1              |
| <code>ByteBool</code> | 0             | 255            |
| <code>WordBool</code> | 0             | 65535          |
| <code>LongBool</code> | -2147483648   | 2147483647     |

Zatem dla typu `LongBool` (i tylko dla niego) możliwa jest sytuacja, w której spełnione są jednocześnie trzy warunki:  $X=FALSE$ ,  $Y=TRUE$  i  $Ord(X)>Ord(Y)$  — wówczas (w pewnym sensie)  $FALSE > TRUE$ .

O prawdziwości powyższych wywodów możesz się o tym przekonać, uruchamiając projekt `BoolSpec1` znajdujący się na dołączonym do książki CD-ROM-ie.

## Typy wyliczeniowe

Typ wyliczeniowy (*enumerated type*) stanowi reprezentację zdefiniowanego *ad hoc* zbioru elementów posiadających wspólną pewną cechę; poszczególne elementy identyfikowane są przez unikalne nazwy nie mające poza tym żadnego innego znaczenia merytorycznego. Oto przykłady typów wyliczeniowych:

```
// stan okna
TStanOkna = (Normalne, Zminimalizowane, Zmaksymalizowane);

// strony świata
TKierunek = (wschod, poludnie, zachod, polnoc);

// kolory karciane
TKolorKarty = (trefl, karo, kier, pik)
```

Każda z wartości typu wyliczeniowego jest stałą tegoż typu — zgodnie z powyższymi definicjami nazwa `karo` jest stałą typu `TKolorKarty`.

Funkcja `Ord` zastosowana do elementu typu wyliczeniowego zwraca numer kolejny tego elementu w definicji typu począwszy od zera — tak więc  $Ord(trefl) = 0$ ,  $Ord(zachod) = 2$  itd. Funkcje `Low` i `High` zwracają (odpowiednio) pierwszy i ostatni element w definicji typu. A zatem, na przykład,  $Low(TStanOkna) := Normalne$ , a  $High(TKolorKarty) = pik$ .

Począwszy od Delphi 6, można w sposób jawny przypisywać poszczególnym elementom (wszystkim lub tylko niektórym) wartość, którą ma dla nich zwrócić funkcja Ord. Oto przykład:

```
TODpowiedz = (Nie=-10, NieWiem=0, Tak=1);
```

Funkcje Low i High zwracają wówczas te elementy, którym przypisano (odpowiednio) najmniejszą i największą wartość.

Elementy, którym nie przypisano wartości w sposób jawny, otrzymują kolejne wartości większe od poprzedników. Jeżeli nie przypisano wartości pierwszemu elementowi, otrzymuje on wartość 0. Zgodnie z poniższą definicją:

```
TPartialEnum = (Dno, Nisko=33, Poziom=0, Krok, Wyzej, Gora=100, Szczyt)
```

zachodzą następujące równości:

```
Ord(Dno)      = 0
Ord(Nisko)    = 33
Ord(Poziom)   = 0
Ord(Krok)     = 1
Ord(Wyzej)    = 2
Ord(Gora)     = 100
Ord(Szczyt)   = 101
```

## Typy znakowe

Wielkość typu znakowego reprezentuje pojedynczy znak alfabetyczny. Aktualnie w Delphi zdefiniowane są dwa fundamentalne typy znakowe:

- ♦ **AnsiChar** — reprezentuje zbiór znaków określony normą ANSI, poszerzony jednak o specyficzne znaki narodowe dla poszczególnych wersji. Wartość tego typu zajmuje jeden bajt, a więc liczba reprezentowanych znaków nie przekracza 256.
- ♦ **WideChar** — reprezentuje znaki wielobajtowe. Aktualnie zmienna typu WideChar zajmuje *dwa bajty*, sam zaś typ odpowiada zbiorowi znaków UNICODE. Pierwsze 256 wartości typu WideChar są identyczne z typem AnsiChar.

Rodzimym typem znakowym Pascala jest typ Char, w dotychczasowych wersjach Delphi równoważny typowi AnsiChar (sytuacja ta może się zmienić w przyszłych wersjach).

## Typy okrojone

Typ okrojony ( *subrange type* ) stanowi *ciągły* podzbiór innego typu porządkowego zwanego *typem bazowym*. Oto przykłady definicji typów okrojonych:

```
TLitery = 'A' .. 'Z'; // typ bazowy - Char
TCyfry = 0 .. 9; // typ bazowy - Integer
TBrakTak = Nie .. NieWiem // typ bazowy - TODpowiedz
```

Typy okrojone dziedziczą z typu bazowego wartości funkcji Ord dla swych elementów, co skądinąd jest całkiem zrozumiałe — Ord('A') równe jest 65 niezależnie od tego, czy znak 'A' postrzegamy jako wartość typu Char czy też typu TLitery.

## Typy zmiennoprzecinkowe

Typy zmiennoprzecinkowe (*floating point types*) reprezentują podzbiory liczb rzeczywistych ze zróżnicowaną dokładnością. Delphi definiuje 6 fundamentalnych typów zmiennoprzecinkowych zgodnych z normą ANSI/IEEE 754. Ich charakterystyka przedstawiona jest w tabeli 2.4.

**Tabela 2.4.** Fundamentalne typy zmiennoprzecinkowe Delphi

| Typ      | Zakres   | Liczba dziesiętnych cyfr znaczących | Dokładność (liczba bitów mantysy) | Rozmiar   |
|----------|--|-------------------------------------|-----------------------------------|-----------|
| Single   | $1.5 \times 10^{-45} \dots 3.4 \times 10^{38}$ (moduł)     | 7-8                                 | 24                                | 4 bajty   |
| Double   | $5.0 \times 10^{-324} \dots 1.7 \times 10^{308}$ (moduł)   | 15-16                               | 53                                | 8 bajtów  |
| Extended | $3.6 \times 10^{-4951} \dots 1.1 \times 10^{4932}$ (moduł) | 19-20                               | 64                                | 10 bajtów |
| Comp     | -263+1 .. 263 -1   | 19-20                               | 63                                | 8 bajtów  |
| Currency | -922337203685477.5808..922337203685477.5807                | 19-20                               | 63                                | 8 bajtów  |
| Real48   | $2.9 \times 10^{-39} \dots 1.7 \times 10^{38}$             | 11-12                               | 39                                | 6 bajtów  |

Comp jest w tym zestawieniu o tyle nietypowy, iż został on zaliczony do zmiennoprzecinkowych jedynie ze względów historycznych — w Turbo Pascalu wszystko, co miało związek z koprocesorem, zaliczane było do arytmetyki zmiennoprzecinkowej. Pod względem fizycznym wartość typu Comp jest ośmiobajtową liczbą całkowitą ze znakiem, o czym można się przekonać, wykonując poniższą sekwencję:

```

Var
  E: Extended;
  C: Comp;

...

E := 2.33;
C := E;
E := C;

```

Po zakończeniu zmienna E zawiera wartość 2, a nie 2,33. Mimo takiego zachowania typ Comp nie może być traktowany na równi z liczbami całkowitymi — przede wszystkim dlatego, że nie jest typem porządkowym. Wraz z pojawieniem się (w Delphi 4) typu Int64 typ Comp stracił praktycznie swoje znaczenie i zachowany został tylko ze względów zachowania kompatybilności.

Typ Currency używany jest głównie do reprezentowania danych finansowych (stąd nazwa). Mimo iż zaliczono go do typów zmiennoprzecinkowych, jest on tak naprawdę liczbą stałoprzecinkową o ustalonej (równej 4) liczbie miejsc dziesiętnych. Podobnie jak typ Comp, jest on reprezentowany w postaci 64-bitowej liczby całkowitej ze znakiem, przy czym zapisywana wartość jest 10 000 razy większa od wartości faktycznie reprezentowanej (stąd właśnie cztery miejsca po przecinku).

Typ `Real48` to dziedzictwo wczesnych wersji Turbo Pascala, w których był *jedynym* typem używanym do obliczeń zmiennoprzecinkowych (nosił wówczas nazwę `Real`). Zajmuje 6 bajtów (= 48 bitów, stąd nazwa `Real48`), format jego reprezentacji binarnej jest całkowicie odmienny od pozostałych typów, zaś wszelkie operacje z jego udziałem odbywają się wyłącznie w sposób programowy, bez jakiegokolwiek wsparcia sprzętowego, co decyduje o niskiej efektywności obliczeń. Został zachowany jedynie ze względów kompatybilności wstecz.

Rodzimy typ zmiennoprzecinkowy Delphi nosi nazwę `Real`. Jeszcze w Delphi 3 był on równoważny temu, co dzisiaj kryje się pod nazwą `Real48`. Począwszy od Delphi 4 stał się on równoważny typowi `Double`, zaś *spadkowy* typ `Real` przechrzczony został na `Real48`. W przypadku uzasadnionej nostalgii można przywrócić typowi `Real` dawne znaczenie, używając dyrektywy kompilacji `{$REALCOMPATIBILITY ON}`.

Oprócz konkretnych wartości liczbowych w ramach typów zmiennoprzecinkowych (z wyjątkiem typu `Real48`) przechowywać można wartości oznaczające brak konkretnej liczby (tzw. nieliczby — ang. NaN = *Not A Number*) jak również symbol oznaczający nieskończoność. Szczegóły dotyczące wewnętrznej reprezentacji typów zmiennoprzecinkowych dostępne są w systemie pomocy Delphi 7 pod hasłem *Real types*.



Jak widać z tabeli 2.4, największą dokładność obliczeń zapewnia typ `Extended` — jest to dokładność, z jaką przechowywane są przez procesor pośrednie wyniki wykonywanych przez niego obliczeń. Niektóre programy mogą jednak sztucznie „obniżyć” tę domyślną dokładność, głównie ze względów zgodności ze starszymi systemami obliczeń zmiennoprzecinkowych. I tak na przykład niektóre procedury `Win32 API` przełączają procesor w tryb dokładności 53-bitowej, charakterystycznej dla typu `Double`. Z tego względu zalecanym typem dla obliczeń zmiennoprzecinkowych w aplikacjach `Win32` jest typ `Double` — i nieprzypadkowo to on właśnie jest rodzimym typem zmiennoprzecinkowym, kryjącym się pod synonimem `Real`.