

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

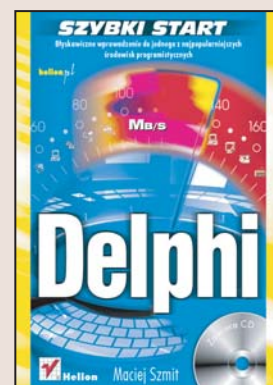
ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Delphi. Szybki start

Autor: Maciej Szmit
ISBN: 83-246-0226-7
Format: B5, stron: 224



Język Pascal, kojarzący się głównie z nauką programowania, stał się podstawą jednego z najpopularniejszych obecnie środowisk programistycznych — Delphi firmy Borland. To graficzne środowisko, pozwalające na szybkie tworzenie aplikacji dzięki możliwości składania ich z „klocków” zwanych komponentami, błyskawicznie zyskało uznanie programistów na całym świecie. Kolejne wersje Delphi oferowały coraz większe możliwości. Jego najnowsza wersja pozwala na tworzenie programów dla platformy .NET i korzystanie nie tylko z języka Pascal, ale również z coraz popularniejszego C#.

Książka „Delphi. Szybki start” to doskonały przewodnik po Delphi dla początkujących użytkowników. Dzięki niemu poznasz język Object Pascal i zasady programowania obiektowego. Nauczysz się wykorzystywać komponenty do tworzenia aplikacji i od zera napiszesz własne programy. Dowiesz się, czym się różni Delphi 7 od Delphi 2005, i wykorzystasz możliwości obu tych środowisk.

- Interfejs użytkownika środowiska Delphi
- Elementy języka Object Pascal
- Programowanie obiektowe
- Obsługa wyjątków
- Tworzenie okien dialogowych
- Projektowanie interfejsów użytkownika
- Korzystanie z komponentów

Poznaj możliwości środowiska Delphi



Spis treści

	Wprowadzenie	7
Rozdział 1.	Zaczynamy	13
	Interfejs użytkownika	13
	Pierwszy program w Delphi	17
	Aplikacje konsolowe	18
	Diagramy składniowe	25
Rozdział 2.	Nieobiektowe elementy języka Object Pascal	27
	Stałe, identyfikatory i wyrażenia	27
	Typy danych, zmienne, instrukcja przypisania, rzutowanie typów, funkcje ord, pred i succ	29
	Zmienne z wartością początkową	37
	Operatory	38
	Operacje wejścia-wyjścia, procedury write/writeln i read/readln	42
	Typy łańcuchowe i operator konkatencji	43
	Instrukcja pusta i instrukcja złożona	44
	Instrukcja warunkowa	45
	Instrukcja wyboru	49
	Definiowanie własnych typów, typy wyliczeniowe i okrojone, zgodność typów, zgodność w sensie przypisania	50
	Typ wariantowy	54
	Definicja statycznego typu tablicowego	55
	Instrukcja iteracyjna, procedury break i continue	56
	Instrukcja iteracyjna z warunkiem na końcu	59
	Instrukcja iteracyjna z warunkiem na początku	60
	Typ rekordowy (bez wariantów), nazwy kwalifikowane i instrukcja wiążąca	62
	Rekordy z wariantami	66

Podprogramy — pojęcia podstawowe: funkcje, procedury, zmienne lokalne, parametry formalne i aktualne, dyrektywy języka, parametry o domyślnej wartości, procedury inc i dec	69
Podprogramy — efekty uboczne	76
Podprogramy — rekurencja	77
Podprogramy — przeciążanie	78
Podprogramy — konwencje wywołania i dyrektywa forward	79
Śledzenie działania programu	81
Typ zbiorowy, operator in	83
Typy wskaźnikowe i zmienne dynamiczne	85
Dynamiczny typ tablicowy	91
Tablice otwarte i wariantowe tablice otwarte	92
Moduły i przestrzenie nazw	95
Typ plikowy	98
Typ proceduralny	103
Etykiety i instrukcja skoku	104
Procedury kierujące działaniem programu: exit, halt, runerror, sleep, abort	105
Operator @ i funkcja addr	106

Rozdział 3.

Wprowadzenie do technik obiektowych.

Wybrane obiektowe elementy języka Object Pascal 109

Klasy, obiekty, metaklasy, generalizacja	110
Polimorfizm, metody dynamiczne i wirtualne	115
Abstrakcja	119
Hermetyzacja	122
Agregacja	125
Asocjacja i porozumiewanie się za pomocą komunikatów	125
Operatory is i as	126
Interfejsy (typ interface)	129
Podstawy obsługi sytuacji wyjątkowych	132
Programowanie z użyciem wątków	139
Przeciążanie operatorów w Delphi 2005 dla .NET	143

Rozdział 4.	Programowanie wizualno-obiektowe	145
	Pierwsza aplikacja okienkowa	145
	Implementacja prostych operacji wejścia i wyjścia w programie okienkowym. Komponent TEdit, typ Pchar, modalne i niemodalne okna dialogowe	154
	Tworzenie dodatkowych okien w programie	161
	Przetwarzanie komunikatów Windows. Metoda ProcessMessages obiektu Application	163
	Dynamiczne tworzenie komponentów	165
	Biblioteki DLL w Windows. Dyrektywa external	169
	Pakiety	173
	Instalacja i deinstalacja dodatkowych komponentów	175
	Tworzenie własnych komponentów	181
	Zakończenie	191
Dodatek A	Bibliografia	193
Dodatek B	Identyczność typów, zgodność i zgodność w sensie przypisania	195
Dodatek C	Słowa kluczowe i symbole specjalne	197
Dodatek D	Dyrektywy języka w Delphi 2005	199
Dodatek E	Spis rysunków	201
Dodatek F	Spis tabel	205
Dodatek G	Spis listingów	207
	Skorowidz	213

Stałe, identyfikatory i wyrażenia

Język Delphi operuje pojęciami wyrażeń i zmiennych w sposób podobny do innych języków programowania. Pojęcie **stałej** jest używane w znaczeniu matematycznym (na przykład stała e — podstawa logarytmów naturalnych odpowiadająca liczbie równej w przybliżeniu 2,718281828). W programie stała będzie po prostu wartością reprezentowaną przez jakiś symbol (nazwę stałej). Ilekroć w programie (w kodzie źródłowym) wystąpi nazwa stałej, Delphi podstawí w tym miejscu odpowiednią wartość. Nazwa stałej nazywana jest jej **identyfikatorem**. Oczywiście musisz poinformować Delphi, że pod daną nazwą będzie kryła się jakaś wartość. Sam z siebie program nie będzie wiedział, czy symbol „X” oznacza wartość pięć, czy może osiem. Definicje stałych zapisujemy również w części biernej programu, poprzedzając je słowem `const` (ang. *constant* — stała).

Wyrażenie jest zbiorem wartości stałych, zmiennych i łączących je operatorów (czyli symboli oznaczających operacje, na przykład operacje dodawania oznaczamy operatorem „+”). W obrębie wyrażeń możesz również używać nawiasów okrągłych. Na przykład wyrażeniem jest $22+432+23-2098$ albo $88+(23-23)$.

Do wyświetlania na ekranie wartości wyrażeń, stałych i zmiennych służą w języku Pascal (i w Delphi) wspomniane wcześniej procedury: `write`, wyświetlająca wyrażenie podane jako parametr, i `writeln`, która wyświetla wyrażenie podane jako parametr i przesuwa kursor do następnej linii.

Wskazówki

- Parametry procedur podajemy w Delphi w nawiasach okrągłych.
- Procedura `writeln` wyświetla obliczoną uprzednio wartość podanego jako parametr wyrażenia.

Na listingu 2.1 przedstawiony został przykład programu wykorzystującego procedurę `writeln`.

Na listingu 2.2 przedstawiono program, w którym użyto stałej o nazwie „dwa”, zdefiniowanej w sekcji definicji stałych (linijki 3 i 4). Jak widzisz, do poinformowania programu o wartości stałej użyto znaku równości. Efektem działania programu jest wyświetlenie na ekranie liczby 5, która jest wynikiem obliczenia wartości wyrażenia będącego parametrem wywołania procedury `writeln`.

Wskazówka

- Zwróć uwagę, że po słowie kluczowym `const` nie ma średnika. Podobnie średnika nie używamy po innych słowach kluczowych oznaczających początki sekcji.

Możemy w tym miejscu wyjaśnić pojęcie definicji stałej, posługując się poznanymi uprzednio diagramami składniowymi (rysunek 2.1).

W definicji stałych możliwe jest dodatkowo użycie niektórych funkcji wbudowanych¹, tak aby stała (lub zmienna inicjowana) przyjmowała wartość wyrażenia zbudowanego za ich pomocą, na przykład:

```
{1} const
{2}   L:longint=trunc(9.3212);
{3}   M =trunc(9.3212);
```

Można również użyć rzutowania typów (pojęcie to zostanie wyjaśnione w kolejnym podrozdziale).

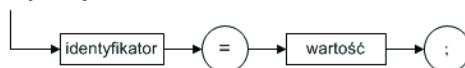
Listing 2.1. Program obliczający wartość wyrażenia $2+2$ i wyświetlający wynik

```
Listing
{1} program Project1;
{2} {$APPTYPE CONSOLE}
{3} begin
{4}   writeln(2+2);
{5}   readln;
{6} end.
```

Listing 2.2. Przykład użycia stałej

```
Listing
{1} program Project1;
{2} {$APPTYPE CONSOLE}
{3} Const //tu zaczyna sie sekcja definicji stałych
{4}   dwa=2; //definicja stałej o nazwie „dwa”
           równej 2
{5} begin
{6}   writeln(dwa+3);
{7}   readln;
{8} end.
```

Definicja stałej



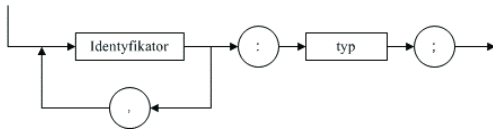
Rysunek 2.1. Diagram składniowy pojęcia „definicja stałej” (wersja uproszczona)

Wskazówka

- Pamiętaj: jeśli chcielibyśmy, żeby nasz program wyświetlił na ekranie nie liczbę, a napis (ciąg znaków), powinniśmy jako parametru wywołania procedury `writeln` użyć tego ciągu ujętego w apostrofy.

¹ Funkcje: `abs`, `chr`, `hi`, `high`, `length`, `lo`, `low`, `odd`, `ord`, `pred`, `round`, `sizeof`, `succ`, `swap` oraz `trunc`. Pojęcie funkcji zostanie wyjaśnione w dalszej części książki.

Deklaracja zmiennych



Rysunek 2.2. Diagram składniowy pojęcia „deklaracja zmiennych”

Typy danych, zmienne, instrukcja przypisania, rzutowanie typów, funkcje ord, pred i succ

Jak zapewne się domyślasz, oprócz stałych bardzo przydałyby się w programie struktury danych, których wartości zmieniałyby się w trakcie działania programu. Takimi strukturami są **zmienne**. Zmienna to struktura danych o określonej nazwie, służąca do przechowywania wartości określonego typu, która ta wartość może zmieniać się w trakcie działania programu. Pojęcie to jest używane podobnie jak pojęcie zmiennej w matematyce. Jeśli mówimy np. o liczbie X , to przez ten symbol rozumiemy jakąś dowolną ustaloną liczbę, na której będziemy przeprowadzać rozmaite operacje. Deklaracje zmiennych zapisujemy w części biernej programu w **sekcji deklaracji zmiennych**, poprzedzając je słowem `var` (ang. *variables* — zmienne).

Każda zmienna musi należeć do określonego typu, to znaczy decydując się na użycie w programie zmiennej, musimy zdecydować, czy będzie ona na przykład liczbą (i jakiego rodzaju), czy napisem. W przeciwnym razie wyniki działania naszego programu nie byłyby jednoznacznie określone. Na przykład z połączenia napisów (znaków pisma) „1” i „0” otrzymamy napis „10”, natomiast dodanie liczb 1 i 0 da w wyniku oczywiście jeden. Dlatego też deklaracja zmiennej zawiera określenie typu, do którego zmienna należy, przy czym przez **typ danych** rozumie się zbiór możliwych wartości wraz z określonym identyfikatorem (rysunek 2.2).

W Delphi istnieje rozbudowana hierarchia typów, z których najważniejsze są typy proste (rzeczywiste i porządkowe), strukturalne (tablicowe, rekordowe, zbiorowe, plikowe i obiektowe), wskaźnikowe, wariantowe, proceduralne oraz łańcuchowe. Najprostsze typy, z którymi będziesz się spotykać, to typ znakowy deklarowany słowem `char` (ang. *character* — znak pisma), oznaczający pojedynczy znak pisma (literę, cyfrę, znaki przestankowe itd.), typ łańcuchowy `string` (ang. *string* — sznurek), używany do zapisu łańcucha znaków, typ całkowitoliczbowy `integer` (ang. *integer* — całkowitoliczbowy), używany do zapisu liczb całkowitych (dodatnich i ujemnych) dających się zapisać w dwóch bajtach pamięci (wliczając bit znaku), oraz typ rzeczywisty `real` (ang. *real* — rzeczywisty), używany do zapisu liczb posiadających część ułamkową.

Wskazówka

- Mówimy o sekcji definicji stałych (gdyż tam definiujemy stałe — podajemy zarówno ich identyfikatory, jak i wartości) i o sekcji deklaracji zmiennych (tam bowiem deklarujemy jedynie, że w programie będziemy używać zmiennych o określonych nazwach należących do określonych typów).

Do nadania zmiennej określonej wartości służy **instrukcja przypisania**, zapisywana za pomocą symbolu dwukropka i znaku równości `:=` (na przykład zapis `X:=8` czytamy: „niech zmienna X przyjmie wartość osiem”).

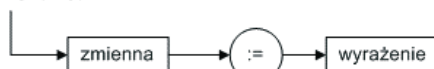
W przykładzie przedstawionym na listingu 2.3 widzimy zadeklarowaną (linia 4) zmienną całkowitoliczbową o nazwie `x`, której w części czynnej programu za pomocą instrukcji przypisania (linia 6) przypisujemy wartość osiem, a następnie wyświetlamy tę wartość na ekranie (linia 8).

Formalna definicja pojęcia „instrukcja przypisania” przedstawiona jest na rysunku 2.3.

Listing 2.3. Wykorzystanie zmiennej

```
Listing
{1}  program Project2;
{2}  {$APPTYPE CONSOLE}
{3}  var
{4}    x:integer;
{5}  begin
{6}    x:=8;
{7}    writeln('wartosc zmiennej xks
      wynosi:');
{8}    writeln(x);
{9}    readln;
{10} end.
```

instrukcja przypisania



Rysunek 2.3. Definicja instrukcji przypisania

Listing 2.4. Program obliczający objętość kuli o promieniu 2

```

Listing
{1}  program Project1;
{2}  {$APPTYPE CONSOLE}
{3}  {program oblicza objetosc kuli o promieniu
      rownym 2}
{4}  var
{5}    promien, objetosc:real; {dwie zmienne
      rzeczywiste}
{6}  begin
{7}    promien:=2; {przypisz zmiennej promien
      wartosc dwa}
{8}    objetosc:=4*3.14*promien*
      ➡promien*promien/3;
{9}    writeln('Objetosc kuli o promieniu 2
      wynosi');
{10}   writeln(objetosc);
{11}   writeln('Nacisnij ENTER');
{12}   readln;
{13} end.

```

W kolejnym przykładzie (listing 2.4) przedstawiono program obliczający objętość kuli o promieniu 2.

W sekcji deklaracji zmiennych zadeklarowano dwie zmienne o nazwach *promien* i *objetosc*, obie typu *real* (linia 5). W części czynnej programu (po słowie kluczowym *begin*) zmiennej *promien* przypisano wartość 2, natomiast zmiennej *objetosc* wartość

wyrażenia $\frac{4}{3}\pi \cdot (\text{promien})^3$ które wyraża wzór na objętość kuli.

- Do oznaczenia operacji mnożenia służy w Delphi znak gwiazdki ***.
- Podnoszenie do sześciennu w przykładzie 2.4 zrealizowano jako trzykrotne mnożenie przez siebie zmiennej *promien*, w języku Pascal nie ma bowiem operatora potęgowania.

Otrzymany na ekranie zapis wyniku: 3.34933333333333E+0001 oznacza liczbę 3,34933333333333 · 10¹.

W języku Pascal separatorem części ułamkowej jest kropka, natomiast zapis E+0001 znaczy „razy dziesięć do potęgi pierwszej” (jest to tak zwany zapis naukowy, litera E jest skrótem angielskiego słowa *exponent*, czyli wykładnik). Objętość kuli wynosi zatem około 33,493 (słownie: trzydzieści trzy i czterysta dziewięćdziesiąt trzy tysięczne).

Typy liczbowe (to jest typy całkowite oraz rzeczywiste) różnią się od siebie między innymi rozmiarem pamięci przeznaczonej na zapisanie wartości danej zmiennej oraz sposobem jej reprezentacji. Jeżeli zadeklarujesz daną zmienną jako należącą na przykład do typu całkowitego `byte`, oznaczać to będzie, że na przechowywanie jej wartości w pamięci RAM komputera zarezerwowano jeden bajt (osiem bitów), zatem największą wartością, jaką zmienna będzie mogła przyjąć, jest 255 (dwójkowo 11111111), a najmniejszą — zero. Próba przypisania tej zmiennej większej wartości może prowadzić do nieoczekiwanych rezultatów działania programu, ponieważ wystąpi tzw. **błąd nadmiaru stałoprzecinkowego** (ang. *Static Point Overflow, SPO*). Z tego też powodu w trakcie tworzenia programu istotny jest właściwy dobór typów dla każdej zmiennej. Efektem działania programu z listingu 2.5. będzie wyświetlenie liczby zero.

Będzie się tak działo dlatego, że reprezentacją liczby typu `byte` jest po prostu wartość zapisana w kolejnych bitach jednego bajta. Dla liczby 255 jest to osiem jedynek:

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

Po zwiększeniu tej wartości o jeden w pamięci zostanie zapisana liczba 256, czyli dwójkowo 10000000, przy czym najbardziej znaczący bit będzie się już znajdował poza zakresem komórki, w której zapisana jest zmienna, i zostanie pominięty:

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

zatem program, wyświetlając jej wartość, uwzględni jedynie osiem zer i wyświetli wartość zero.

Listing 2.5. Przykład błędnego doboru typu zmiennej

```
Listing
{1}  program Project2;
{2}  {$APPTYPE CONSOLE}
{3}  var
{4}      x:byte;
{5}  begin
{6}      x:=255;
{7}      x:=x+1;
{8}      writeln('wartosc zmiennej iks
        wynosi:');
{9}      writeln(X);
{10}     readln;
{11}  end.
```

Nazwy poszczególnych typów liczbowych, rozmiary pamięci i zakresy, jakie mogą przyjmować należące do nich zmienne, przedstawiono w tabelach 2.1 i 2.2.

Wskazówka

- Pamiętaj, że wynikiem dzielenia dwóch liczb całkowitych jest często liczba niecałkowita. Operacja dzielenia dwóch liczb w Pascalu zwraca zawsze wynik typu rzeczywistego (nawet jeżeli dzielnik mieści się w dzielnej całkowitą ilość razy, to jest jeżeli wynik ilorazu jest całkowity), dlatego też jeżeli chcemy wynik dzielenia przypisać jakiejś zmiennej, to musi być to zmienna typu rzeczywistego.

Przez **typy porządkowe** rozumie się grupę typów, w których określony jest pewien porządek, to jest pomiędzy elementami należącymi do tego typu można określić relacje: większości i następstwa. Istnienie relacji następstwa oznacza, że dla elementu takiego typu można podać jego następnika lub poprzednika. Na przykład jednobajtowa liczba całkowita 2 jest większa od liczby całkowitej 0, jej poprzednikiem jest liczba 1, a następnikiem liczba 3.

Tabela 2.1. Typy rzeczywiste

Nazwa typu	Minimum	Maksimum	Liczba cyfr znaczących	Rozmiar w bajtach
Real48	$-2,9 \cdot 10^{39}$	$1,7 \cdot 10^{38}$	11 – 12	6
Single	$-1,5 \cdot 10^{45}$	$3,4 \cdot 10^{38}$	7 – 8	4
Double	$-5,0 \cdot 10^{324}$	$1,7 \cdot 10^{308}$	15 – 16	8
Extended	$-3,6 \cdot 10^{4951}$	$1,1 \cdot 10^{4932}$	19 – 20	10
Comp	$-(2^{63})+1$	$(2^{63})+1$	19 – 20	8
Currency	-922337203685477,5808	922337203685477,5807	19 – 20	8
Real	$-5,0 \cdot 10^{324}$	$1,7 \cdot 10^{308}$	15 – 16	8

Tabela 2.2. Typy całkowite

Nazwa typu	Minimum	Maksimum	Rozmiar w bajtach
Integer	-2147483648	2147483647	32 z bitem znaku
Cardinal	0	4294967295	32 bez bitu znaku
Shortint	-128	127	8 bez bitu znaku
Smallint	-32768	32767	16 z bitem znaku
Longint	-2147483648	2147483647	32 z bitem znaku
Int64	$-(2^{63})$	$2^{63}-1$	64 z bitem znaku
Byte	0	255	8 bez bitu znaku
Word	0	65535	16 bez bitu znaku
Longword	0	4294967295	32 bez bitu znaku

Aby przekonać się, jaki jest numer porządkowy elementu w danym typie (listing 2.6), posługujemy się funkcją `ord(element)`, poprzednika danego elementu zwraca funkcja `pred(element)`, a następnika — `succ(element)`.

Typów określających liczby rzeczywiste nie zaliczamy do typów porządkowych, ponieważ — mimo że wśród liczb rzeczywistych można zdefiniować relację większości — nie można podać dokładnie określonego, jednego bezpośredniego następnika ani poprzednika dowolnej liczby rzeczywistej.

Typami wyliczeniowymi nazywamy te typy liczbowe, które są jednocześnie typami porządkowymi.

W **typie znakowym** (`char`) relacja większości określona jest w ten sposób, że kolejne znaki uporządkowane są według ich kodów ASCII, w szczególności cyfry (od zera do dziewięciu) poprzedzają wielkie litery (w kolejności alfabetycznej), a te z kolei małe litery (w kolejności alfabetycznej).

Ostatnimi z typów wyliczeniowych są **typy logiczne**. Używane są do oznaczenia stanów logicznych: prawda (`true`) oraz fałsz (`false`). W standardowym Pascalu istnieje jeden typ logiczny deklarowany za pomocą słowa `boolean`, w Delphi istnieje kilka typów logicznych, które różnią się reprezentacją (liczbą i sposobem rozumienia bitów przeznaczonych do przechowywania w pamięci wartości typu logicznego). Są to typy `Boolean`, `ByteBool`, `WordBool` oraz `LongBool`. W klasycznym typie `Boolean` do zapamiętania wartości przeznaczony jest jeden bajt pamięci i obowiązuje relacja `False < True`, przy czym następnikiem `False` jest `True`, a `True` nie ma następnika. W pozostałych typach logicznych następnikiem `True` jest `False`, a następnikiem `False` jest `True`. Stosowanie typu `boolean` jest zalecane wszędzie tam, gdzie nie zachodzi konieczność zachowania zgodności z używanymi bibliotekami lub wymogi programu nie nakazują użycia innych typów logicznych.

Listing 2.6. Użycie funkcji `ord`, `pred` i `succ`

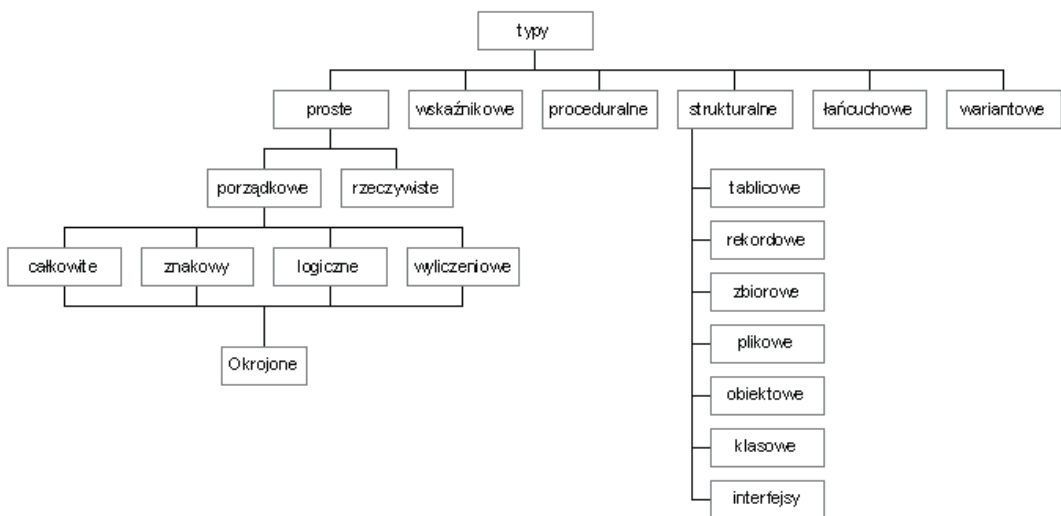
```
Listing
{1} program Project1;
{2} {$APPTYPE CONSOLE}
{3}
{4} begin
{5}   writeln(succ('A')); //jaka litera
                        //następuje po "A"
{6}   writeln(pred('C')); //jaka litera
                        //poprzedza "C"
{7}   writeln(ord(12));  //numer porządkowy
                        //liczby 12 wśród liczb typu byte
{8}   readln;
{9} end.
```

W standardzie języka Pascal każda zadeklarowana zmienna ma nieustaloną wartość początkową, dlatego też zanim zostanie do czegoś użyta, powinna mieć przypisaną jakąś wartość (w przeciwnym razie działanie programu może przynieść nieoczekiwane efekty). Często w początkowej części programu programista umieszcza polecenia **zainicjowania zmiennych**, to jest przypisuje wszystkim zmiennym występującym w programie jakąś wartość początkową (na przykład zero dla zmiennych liczbowych). Delphi standardowo powinno inicjować zadeklarowane zmienne typów prostych:

- ◆ liczby — liczbą zero,
- ◆ znaki — znakiem o numerze porządkowym zero,
- ◆ zmienne typów logicznych — wartością False.

W praktyce jednak inicjowanie to nie zawsze działa poprawnie, dlatego też bezpieczniej jest przeprowadzać je w programie samodzielnie.

Rysunek 2.4 przedstawia hierarchię typów w Delphi.



Rysunek 2.4. Podstawowa hierarchia typów w Delphi

Rzutowanie typów jest to wymuszenie na programie interpretacji zawartości danej komórki pamięci jako wartości należącej do określonego typu. Jeśli na przykład w komórce pamięci o długości jednego bajta zapisaliśmy liczbę 65, to jej zawartość będzie w rzeczywistości zapisana w postaci liczby dwójkowej 01000001. Ten sam zapis i ta sama liczba zinterpretowana jako wartość kodu ASCII odpowiada wielkiej literze A. Jeśli zatem wymusimy na programie taką interpretację, to powinien on potraktować tę wartość jako zakodowaną literę A (listing 2.7). Rzutowanie typów można w Pascalu zrealizować poprzez wywołanie nazwy typu z parametrem (podanym w nawiasach) będącym wartością lub zmienną, która ma być zamieniona na zmienną innego typu.

Możesz również rzutować np. liczbę zero na wartość typu logicznego `boolean`, co da w wyniku wartość `False`. Natomiast w przypadku liczby jeden będzie to wartość logiczna `True` (listing 2.8).

Wskazówki

- Decydując się na użycie rzutowania, musisz dokładnie wiedzieć, w jaki sposób dana wartość jest reprezentowana w pamięci, w przeciwnym razie może dojść do niespodziewanych efektów.
- Jak wspomniano w poprzednim podrozdziale, dopuszczalne jest użycie rzutowania typów w obrębie sekcji definicji stałych, na przykład prawidłowy będzie zapis `Const N=byte(false);`.
- Oprócz rzutowania typów możesz posłużyć się licznymi dostępnymi w standardowych modułach funkcjami konwersji typów, które zazwyczaj są bezpieczniejsze.

Listing 2.7. Rzutowanie typów — interpretacja liczby jako znaku

```
Listing
{1} program Project2;
{2} {$APPTYPE CONSOLE}
{3} begin
{4}   writeln(char(65));
{5}   readln;
{6} end.
```

Listing 2.8. Rzutowanie liczby zero na wartość typu `Boolean`

```
Listing
{1} program Project2;
{2} {$APPTYPE CONSOLE}
{3} begin
{4}   writeln(boolean(0));
{5}   readln;
{6} end.
```

Listing 2.9. Przykład użycia zmiennej inicjowanej

```

Listing
{1}  program Project2;
{2}  {$APPTYPE CONSOLE}
{3}  const
{4}    dziwna_stala:byte=3;
{5}  begin
{6}    writeln(dziwna_stala);
{7}    dziwna_stala:=15;
{8}    writeln(dziwna_stala);
{9}    readln;
{10} end.

```

Zmienne z wartościami początkową

Jak pamiętasz, stałe nie wymagają deklarowania typów, do których należą. Kompilator, dobierając typ stałej, wybiera typy o możliwie najmniejszej liczebności. Programista może umieścić w definicji stałej jawną deklarację typu (na przykład aby wymusić inną jej reprezentację). Na przykład:

```
const liczba_pi:double=3.14;
```

Tak zdefiniowane stałe, zwane **typed constants** — **stałymi o jawnie zadeklarowanym typie**, w rzeczywistości zachowują się jak zmienne (dlatego też nazywa się je czasem **zmiennymi inicjowanymi**). Możliwe jest modyfikowanie ich wartości wewnątrz części czynnej programu.

W kolejnym przykładzie (listing 2.9) w linii 4 w obrębie sekcji definicji stałych zdefiniowano zmienną, której następnie w części czynnej programu (linia 7) przypisano wartość inną niż wartość początkowa.

Operatory

Poznałeś już operatory: + (plus), oznaczający dodawanie, oraz * (razy), oznaczający mnożenie. Wszystkie operatory liczbowe przedstawia tabela 2.3.

Operator `div` służy do dzielenia bez reszty. Wynikiem operacji jest wtedy liczba całkowita (typu całkowitego). W takim przypadku zmienna, w której chcemy zapisać wynik, może być zadeklarowana jako zmienna całkowita, jak pokazano w przykładzie (listing 2.10).

Tabela 2.3. Operatory arytmetyczne dwuargumentowe w Delphi

Operator	Operacja
+ (plus)	Dodawanie
- (minus)	Odejmowanie
* (gwiazdka)	Mnożenie
/ (ukośnik)	Dzielenie
<code>div</code>	Dzielenie bez reszty
<code>mod</code>	Reszta z dzielenia

Listing 2.10. Użycie operatora `div`

```

Listing
{1}  program Project1;
{2}  {$APPTYPE CONSOLE}
{3}
{4}  uses
{5}    SysUtils;
{6}
{7}  var
{8}    a,b,c:byte;
{9}    r:real;
{10} begin
{11}   a:=11;
{12}   b:=5;
{13}   c:=a div b;
{14}   r:=a div b;
{15}   writeln('zapisany w postaci liczby
      całkowitej');
{16}   writeln('wynik dzielenia całkowitego
      11 div 5 wynosi ',c);
{17}   writeln('a zapisany w postaci liczby
      rzeczywistej');
{18}   writeln('wynik tego samego dzielenia
      wynosi ',r);
{19}   writeln('Nacisnij ENTER');
{20}   readln;
{21} end.
```