helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

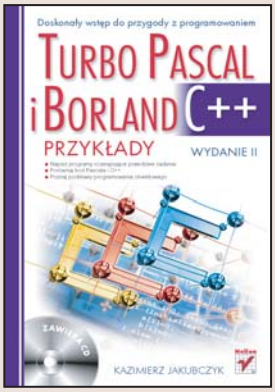
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Turbo Pascal i Borland C++. Przykłady. Wydanie II



Autor: Kazimierz Jakubczyk
ISBN: 83-246-0346-8
Format: B5, stron: 376

Środowiska programistyczne Turbo Pascal i Borland C++ mimo swojego wieku nadal są powszechnie wykorzystywane podczas nauki podstaw programowania. Oczywiście do nauki można również wykorzystać dostępne obecnie doskonałe środowiska wizualne, lecz podstawy języka łatwiej jest opanować, posługując się prostszym narzędziem. Turbo Pascal i Borland C++ są proste w obsłudze, doskonale udokumentowane i wspaniale nadają się do pierwszego kontaktu z programowaniem.

Książka „Turbo Pascal i Borland C++. Przykłady. Wydanie II" jest przewodnikiem po metodach rozwiązywania rzeczywistych problemów matematycznych i fizycznych za pomocą języków Pascal i C++. Implementacja znanych algorytmów jest doskonałym sposobem nauki programowania, a analiza przykładów napisanych zarówno w Pascalu, jak i w C++ może okazać się nieocenioną pomocą dla tych, którzy opanowali jeden z tych języków i chcieliby opanować drugi.

- Operacje na liczbach i datach
- Programowanie grafiki
- Proste animacje
- Implementacja list jednokierunkowych
- Programowanie obiektowe
- Tworzenie programów z wykorzystaniem Windows API

Poznaj podstawy programowania

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

Spis treści

Wstęp **9**

Rozdział 1. Operacje na liczbach **13**

Ile cyfr ma liczba? 13
- Program LCyf w Pascalu 13
- Pętle w Pascalu 14
- Pętle w C i C++ 16
- Program LCyf w C++ 17
- Uruchamianie programu w środowisku programowania 17

Odwrócenie kolejności bitów liczby 18
- Program Bity w Pascalu 18
- Program Bity w C++ 19
- Wyrażenia przypisujące w C i C++ 20

Największy wspólny dzielnik dwu liczb całkowitych 20
- Algorytm Euklidesa 21
- Program Euklides w Pascalu 22
- Program Euklides w C++ 22

Reprezentacja zmiennopozycyjna liczby rzeczywistej 23
- Program RepZmPoz w Pascalu 23
- Program RepZmPoz w C++ 24

Zapis liczby w notacji rzymskiej 25
- Program LRzym w Pascalu 26
- Program LRzym w C++ 27
- Wskaźniki a łańcuchy w C i C++ 28

Współczynniki Newtona i trójkąt Pascala 28
- Algorytm iteracyjny 29
- Algorytm rekurencyjny 30
- Program TPascala w Pascalu 30
- Program TPascala w C++ 31

Tablicowanie funkcji Bessela 32
- Algorytm obliczania sumy szeregu liczbowego 33
- Program FBessela w Pascalu 34
- Program FBessela w C++ 35
- Formatowanie wydruku w funkcji printf 36

Ćwiczenia 37

Rozdział 2. Zadania z kalendarzem ... 39
Dzień tygodnia i numer dnia w roku ... 40
Algorytmy kalendarzowe ... 40
Moduł obliczeniowy Kal w Pascalu ... 41
Program Dzien w Pascalu ... 43
Moduł obliczeniowy Kal w C++ ... 43
Program Dzien w C++ ... 45
Kalendarz miesięczny ... 46
Moduł Klaw czytania znaku w Pascalu ... 46
Moduł Klaw czytania znaku w C++ ... 47
Program KMies w Pascalu ... 48
Program KMies w C++ ... 50
Kolorowanie kalendarza ... 52
Algorytm Gaussa wyznaczania daty Wielkanocy ... 53
Program KMies2 w Pascalu ... 54
Program KMies2 w C++ ... 56
Kalendarz roczny ... 60
Program KRocz w Pascalu ... 60
Wyprowadzanie zmiennej liczby spacji w C ... 61
Program KRocz w C++ ... 62
Tryby dostępu do pliku w języku C ... 63
Drukowanie tekstu na drukarce ... 64
Operacje wejścia-wyjścia dla plików w C++ ... 64
Program KRocz2 w C++ ... 65
Ćwiczenia ... 67

Rozdział 3. Grafika ... 69
Gra w chaos ... 69
Algorytm gry w chaos ... 70
Program Chaos w Pascalu ... 70
Program Chaos w C++ ... 72
Biblioteka graficzna WinBGI ... 73
Wielokąt foremny i gwiazdka ... 74
Wyznaczanie wierzchołków wielokąta foremnego ... 74
Rysowanie wielokąta foremnego w Pascalu ... 75
Rysowanie wielokąta foremnego w C++ ... 76
Wyznaczanie wierzchołków gwiazdki równoramiennej ... 76
Program Gwiazdka w Pascalu ... 77
Program Gwiazdka w C++ ... 78
Najmniejszy wielokąt wypukły ... 79
Algorytm wyznaczania brzegu najmniejszego wielokąta wypukłego ... 80
Program WielWyp w Pascalu ... 81
Program WielWyp w C++ ... 83
Wskaźniki a tablice w C i C++ ... 85
Wyrażenia warunkowe w C i C++ ... 86
Program WielWyp2 w C++ ... 86
Częstotliwość występowania liter w pliku ... 88
Program Litery w Pascalu ... 88
Program Litery w C++ ... 91
Konwersja znaku na łańcuch w C i C++ ... 92
Definiowanie stałych symbolicznych w C++ ... 93
Program Litery2 w C++ ... 93

Liniowa aproksymacja średniokwadratowa ... 94
Przekazywanie parametrów przez wartość i zmienną w Pascalu ... 96
Moduł obliczeniowy LApr w Pascalu ... 97
Moduł obliczeniowy LApr w C++ ... 98
Program Aproks w Pascalu ... 100
Program Aproks w C++ ... 102
Program Aproks2 w C++ ... 103
Wykres funkcji drgań harmonicznych tłumionych ... 104
Okno i widok przy tworzeniu wykresu funkcji ... 105
Program Drgania w Pascalu ... 106
Nazwa funkcji parametrem podprogramu w Pascalu ... 109
Nazwa funkcji a wskaźnik w C i C++ ... 110
Program Drgania w C++ ... 110
Krzywe Sierpińskiego ... 112
Program KrzSierp w Pascalu ... 113
Program KrzSierp w C++ ... 115
Ćwiczenia ... 116

Rozdział 4. Animacja ... 119
Piłeczka ... 119
Program Pileczka w Pascalu ... 120
Program Pileczka w C++ ... 121
Własna funkcja delay ... 122
Wieże Hanoi ... 122
Reprezentacja danych w Pascalu ... 123
Wizualizacja krążków na ekranie monitora ... 124
Nakładanie krążka na szczyt wieży ... 124
Zdejmowanie krążka ze szczytu wieży ... 125
Algorytm rekurencyjny przenoszenia wież ... 126
Program WHanoi w Pascalu ... 126
Program WHanoi w C++ ... 129
Krzywe Lissajous ... 131
Rysowanie krzywej na ekranie monitora ... 131
Odwracający tryb rysowania ... 132
Program Lissa w Pascalu ... 132
Program Lissa w C++ ... 134
Przekazywanie parametrów przez wartość i referencję w C++ ... 136
Program Lissa2 w C++ ... 136
Układ planetarny ... 137
Model komputerowy układu planetarnego ... 138
Program Grawit w Pascalu ... 140
Program Grawit w C++ ... 143
Hipocykloida ... 145
Obliczanie współrzędnych punktów ... 145
Algorytm animacji oparty na kopiowaniu fragmentów obrazu ... 146
Dynamiczne przydzielanie i zwalnianie pamięci ... 147
Program Hipo w Pascalu ... 148
Program Hipo w C++ ... 149
Elementy charakterystyczne dla C++ ... 151
Wyświetlanie pliku rekordów ... 151
Moduł definiujący strukturę pliku w Pascalu ... 152
Moduł definiujący strukturę pliku w C++ ... 153
Tworzenie pliku rekordów ... 154
Program Lista w Pascalu ... 156

Program Lista w C++ ... 159
Program Lista2 w C++ ... 162
Strumieniowe formatowanie liczb rzeczywistych ... 165
Parametry wywołania programu ... 166
Ćwiczenia ... 168

Rozdział 5. Listy jednokierunkowe ... 171
Sito Eratosthenesa ... 172
Definiowanie listy w Pascalu ... 172
Wstawianie elementu na początek listy ... 173
Przeglądanie listy i usuwanie elementów ... 174
Program SitoEra w Pascalu ... 175
Definiowanie i obsługa listy w C++ ... 177
Program SitoEra w C++ ... 177
Wskaźnik NULL w roli wyrażenia warunkowego ... 179
Rozwinięcie dziesiętne ułamka ... 179
Tablica czy lista? ... 179
Generowanie listy cyfr rozwinięcia dziesiętnego ułamka ... 180
Program Rozwin w Pascalu ... 181
Program Rozwin w C++ ... 182
Rozdanie kart do brydża ... 184
Generowanie talii kart i jej tasowanie ... 184
Rozdanie kart czterem graczom ... 185
Wstawianie elementu do listy uporządkowanej ... 185
Program Brydz w Pascalu ... 187
Program Brydz w C++ ... 189
Przekazywanie wskaźnika przez wartość i referencję w C++ ... 192
Zmienne statyczne w C i C++ ... 192
Skorowidz słów pliku tekstowego ... 192
Czytanie słowa w Pascalu ... 193
Lista słów skorowidza z podlistami numerów wierszy ... 194
Aktualizacja listy słów skorowidza ... 194
Program Skorow w Pascalu ... 196
Czytanie słowa w C++ ... 198
Łańcuchy dynamiczne w C++ ... 200
Program Skorow w C++ ... 200
Program Skorow2 w C++ ... 203
Porządkowanie pliku rekordów ... 205
Rozpraszanie rekordów pliku ... 205
Program Sort w Pascalu ... 207
Program Sort w C++ ... 210
Program Sort2 w C++ ... 211
Ćwiczenia ... 212

Rozdział 6. Programy obiektowe ... 215
Punkty ... 216
Pierwszy kontakt z typem obiektowym w Pascalu ... 217
Dostęp do składowych obiektu ... 219
Metody wirtualne, konstruktor i destruktor ... 219
Moduł Punkty w Pascalu ... 220
Klasa w C++ i jej moduł definiujący ... 222
Moduł Punkty w C++ ... 223
Moduł Ruch przesuwania punktu w Pascalu ... 225
Moduł Ruch przesuwania punktu w C++ ... 226

Program przesuwania punktu w Pascalu ... 226
Obiekty dynamiczne w Pascalu ... 227
Wywoływanie konstruktorów i destruktorów w C++ ... 228
Program przesuwania punktu w C++ ... 228
Okręgi ... 230
Moduł Okregi w Pascalu ... 230
Program przesuwania okręgu w Pascalu ... 231
Moduł Okregi w C++ ... 232
Program przesuwania okręgu w C++ ... 234
Łamane ... 234
Moduł Lamane w Pascalu ... 235
Program przesuwania prostokąta w Pascalu ... 236
Moduł Lamane w C++ ... 237
Program przesuwania prostokąta w C++ ... 239
Program przesuwania hipocykloidy w Pascalu ... 239
Program przesuwania hipocykloidy w C++ ... 241
Lista figur geometrycznych ... 242
Moduł Figury w Pascalu ... 242
Program Pojazd w Pascalu ... 244
Moduł Figury w C++ ... 245
Program Pojazd w C++ ... 247
Fontanna ... 248
Koncepcja typu potomnego reprezentującego kroplę wody ... 248
Program Fontanna w Pascalu ... 249
Program Fontanna w C++ ... 252
Osoby ... 254
Moduł Osoby w Pascalu ... 254
Program tworzący przykładową listę pracowników w Pascalu ... 256
Moduł Osoby w C++ ... 259
Program tworzący przykładową listę pracowników w C++ ... 260
Edycja wiersza tekstu ... 262
Tworzenie typu obiektowego edycji łańcucha w Pascalu ... 263
Przesyłanie danych do edycji i udostępnianie wyniku edycji ... 264
Programowanie operacji edycyjnych ... 265
Moduł Edycja w Pascalu ... 266
Funkcje edycji łańcucha i listy łańcuchów ... 270
Moduł Edycja w C++ ... 271
Edycja pola daty w Pascalu ... 276
Program Plik edycji danych osobowych w Pascalu ... 278
Edycja pola daty w C++ ... 281
Program Plik edycji danych osobowych w C++ ... 283
Program Plik2 edycji danych osobowych w C++ ... 284
Ćwiczenia ... 286

Rozdział 7. Programowanie w Windows ... 289
Pierwszy program w Windows ... 290
Nowe typy danych ... 292
Funkcje API Windows ... 294
Funkcja główna programu ... 294
Procedura okna ... 297
Malowanie okna ... 298
Generowanie fraktali ... 299
Menu i inne zasoby programu ... 301
Program Fraktale ... 303

Przechwytywanie komunikatów menu ... 307
Procedura okna dialogowego ... 308
Rysowanie gwiazdki ... 309
Rysowanie wypełnionych obszarów ... 310
Okno dialogowe z kontrolkami edycyjnymi ... 311
Program Gwiazdka ... 312
Tworzenie i wybieranie obiektów graficznych ... 316
Praca z kontrolkami w oknie dialogowym ... 318
Rotacja gwiazdki ... 319
Tworzenie i malowanie bitmap ... 319
Komunikaty zegarowe ... 321
Aktywne i nieaktywne elementy menu ... 321
Program Rotacja ... 322
Spacerujący skrzat ... 327
Operacje rastrowe ... 327
Klasa animacji Sprite ... 329
Bitmapy w zasobach programu ... 332
Program Skrzat ... 333
Przesuwanie muchy ... 337
Komunikaty klawiaturowe ... 337
Komunikaty myszy ... 338
Program Mucha ... 338
Liczby rzymskie ... 344
Definicja okna dialogowego ... 345
Program Rzymskie ... 346
Ćwiczenia ... 350

Dodatek A Darmowe narzędzia do programowania w C++ ... 353

Literatura ... 357

Skorowidz ... 359

Rozdział 6.

Programy obiektowe

Niniejszy rozdział stanowi wprowadzenie do programowania obiektowego w językach Turbo Pascal i Borland C++. Podstawowe zagadnienia, takie jak definiowanie typów obiektowych, dziedziczenie i polimorfizm, określanie praw dostępu do składowych obiektu, tworzenie listy dynamicznej obiektów oraz pisanie kodu modułu nadającego się do dalszego wykorzystania i rozszerzania, są omawiane w trakcie tworzenia przykładowych programów dotyczących:

- wyświetlania i przesuwania na ekranie punktu, okręgu i linii łamanej,
- tworzenia i wykreślania figur złożonych z okręgów, łuków i łamanych,
- przedstawienia fontanny w postaci listy zbudowanej z różnokolorowych okręgów imitujących poruszające się kropelki wody,
- wykorzystania listy obiektów do przechowywania danych grupy osób,
- edycji wiersza tekstu,
- edycji pliku rekordów.

Tradycyjne podejście do programowania, zwane **programowaniem proceduralnym**, cechuje rozdzielenie dwóch elementów programu: danych i przetwarzających je procedur. Ma ono dwie zasadnicze wady. Po pierwsze, odizolowanie danych od wykonywanych na nich operacji może prowadzić do przypadkowej, niepożądanej zmiany tych danych z powodu braku ich ochrony przed niepowołanym dostępem. Po drugie, dostosowanie opracowanego tą metodą oprogramowania do nowych warunków wymaga modyfikacji kodu źródłowego, co często jest bardzo trudne, a niekiedy z powodu jego niedostępności niemożliwe.

Inaczej przedstawia się sytuacja w przypadku **programowania zorientowanego obiektowo** (ang. *Object Oriented Programming* — OOP), zwanego krócej **programowaniem obiektowym**. Charakteryzuje się ono następującymi cechami:

- **hermetyzacja** (kapsułkowanie, ang. *encapsulation*) — definiowanie nowego typu danych, zwanego obiektowym, jednoczącego pola reprezentujące dane oraz działające na nich procedury i funkcje, zwane **metodami**;

- **dziedziczenie** (ang. *inheritance*) — definiowanie nowego typu obiektowego, zwanego potomnym lub pochodnym, na podstawie typu istniejącego, zwanego bazowym, dziedziczącego pola i metody typu bazowego;
- **polimorfizm** (ang. *polymorphism* — wielopostaciowość) — definiowanie tzw. **metod wirtualnych**, które w typie potomnym i bazowym mają tę samą nazwę, ale określają różne akcje.

Definicja typu obiektowego, zwanego w nowszej wersji Pascala[1] i w C++ **klasą**, określa pola, podobnie jak w przypadku rekordu lub struktury, oraz metody obróbki przechowywanych w nich danych. Hermetyzacja — łącząca pola i metody — zawiera mechanizmy pozwalające na kontrolowanie dostępu do pól i metod obiektu dla pozostałych części programu. Z kolei dziedziczenie umożliwia tworzenie nowych typów obiektowych na podstawie typów istniejących. Typy pochodne automatycznie dziedziczą właściwości typów bazowych, a ponadto mogą zawierać nowe pola i metody rozszerzające możliwości odziedziczone po typach bazowych. Natomiast polimorfizm, stanowiący uzupełnienie dziedziczności, pozwala na określanie innego kodu metod w typach pochodnych. Oznacza to, że ta sama metoda może w odniesieniu do obiektu pochodnego wywołać inne działanie niż w przypadku obiektu bazowego.

Dziedziczenie i polimorfizm sprawiają, że możliwe jest tworzenie kodu nadającego się do wykorzystania w warunkach nieprzewidzianych w momencie jego pisania. Kod opracowany obiektowo można bez modyfikowania wielokrotnie wykorzystać, wywodząc z niego nowe typy obiektowe i dostosowując ich właściwości do nowych warunków. Ta wielka zaleta programowania obiektowego pozwala na tworzenie wygodnych do wykorzystania bibliotek obiektowych. Przykładem takiego oprogramowania jest biblioteka Turbo Vision, która umożliwia konstruowanie programów interakcyjnych, komunikujących się z użytkownikiem za pomocą myszki i okien o wyglądzie zbliżonym do tych, jakie występują w środowisku Turbo Pascal 7.0 i Borland C++ 3.1.

Punkty

Podstawową zasadą, którą należy kierować się w programowaniu obiektowym, jest traktowanie typu obiektowego jako odpowiednika obiektu rzeczywistego — przedmiotu lub zjawiska, którego program dotyczy. W definicji typu obiektowego należy więc uwzględnić naturę obiektu fizycznego, jego właściwości i zachowanie. Jeżeli zadaniem programu jest kreślenie i przesuwanie na ekranie prostych figur geometrycznych, to przy określaniu składowych najprostszego typu obiektowego, reprezentującego punkt, trzeba uwzględnić współrzędne ekranowe punktu, jego pokazywanie i ukrywanie oraz przemieszczanie.

[1] Chodzi o Object Pascal, który jest językiem programowania w środowisku Delphi.

Pierwszy kontakt z typem obiektowym w Pascalu

W Turbo Pascalu definicja typu obiektowego przypomina definicję typu rekordowego. Różnica polega na użyciu słowa kluczowego `object` zamiast `record` i zadeklarowaniu, oprócz pól, metod obiektu. Deklaracje metod mają postać nagłówków procedur i funkcji, podobnie jak w części `interface` modułu. Pierwsza wersja definicji typu obiektowego reprezentującego punkt może wyglądać następująco:

```
type
  TPunkt = object
    x, y: integer;
    Wid : Boolean;
    procedure Init(xPoc, yPoc: integer);
    procedure Pokaz;
    procedure Ukryj;
    procedure Rysuj;
    procedure PrzesunDo(xNowe, yNowe: integer);
  end;
```

Pola *x* i *y* określają współrzędne punktu, zaś pole *Wid* — jego status (`true` — punkt widoczny, `false` — niewidoczny). Metoda `Init` inicjalizuje obiekt, określając wartości początkowe jego pól, `Pokaz` wyświetla punkt na ekranie, `Ukryj` ukrywa go, a `PrzesunDo` przesuwa w nowe miejsce. Właściwy kod źródłowy tych metod, umieszczony na zewnątrz definicji typu `TPunkt`, ma postać:

```
procedure TPunkt.Init(xPoc, yPoc: integer);
begin
  x := xPoc;
  y := yPoc;
  Wid := false;
end;

procedure TPunkt.Pokaz;
begin
  Wid := true;
  Rysuj;
end;

procedure TPunkt.Ukryj;
var
  Color: word;
begin
  Color := GetColor;
  SetColor(GetBkColor);
  Wid := false;
  Rysuj;
  SetColor(Color);
end;

procedure TPunkt.Rysuj;
begin
  PutPixel(x, y, GetColor);
end;

procedure TPunkt.PrzesunDo(xNowe, yNowe: integer);
var
```

```
  BylWidoczny: Boolean;
begin
  BylWidoczny := Wid;
  if BylWidoczny then Ukryj;
  x := xNowe;
  y := yNowe;
  if BylWidoczny then Pokaz;
end;
```

Metoda `Init` nadaje wartości początkowe wszystkim polom obiektu. Polom *x*, *y* przypisuje wartości współrzędnych określone w parametrach, a polu *Wid* wartość `false`, co oznacza, że wstępnie punkt jest niewidoczny. Metoda `Pokaz` po ustaleniu, że punkt ma być widoczny, wywołuje metodę `Rysuj`, która rysuje go w kolorze bieżącym. Nieco bardziej złożona jest metoda `Ukryj`, która ukrywa punkt, wymuszając narysowanie go w kolorze tła. Najpierw zapamiętuje w zmiennej pomocniczej kolor bieżący, potem ustawia kolor tła jako kolor bieżący, a po ukryciu punktu przywraca pierwotny kolor bieżący. Metoda `PrzesunDo` przed przesunięciem punktu ukrywa go, gdy jest widoczny, a po przesunięciu pokazuje go z powrotem w nowym miejscu, gdy na początku był widoczny.

Wykorzystanie zdefiniowanego powyżej typu obiektowego jest proste. Zadeklarowanie zmiennej obiektowej *pkt* typu `TPunkt` ma postać zwykłej deklaracji:

```
var
  pkt: TPunkt;
```

Aby utworzyć punkt i pokazać go na ekranie, następnie przesunąć w nowe miejsce i na koniec ukryć, wystarczy posłużyć się metodami obiektu *pkt*:

```
pkt.Init(320, 240);        { Utworzenie niewidocznego punktu }
pkt.Pokaz;                 { Wyświetlenie punktu }
pkt.PrzesunDo(400, 200);   { Przesunięcie punktu }
pkt.Ukryj;                 { Ukrycie punktu }
```

Wewnątrz zakresu instrukcji `with`, podobnie jak w przypadku rekordu, można odwoływać się do składowych obiektu bez poprzedzania ich jego nazwą. Powyższy kod jest więc równoważny następującemu:

```
with pkt do
begin
  Init(320, 240);          { Utworzenie niewidocznego punktu }
  Pokaz;                   { Wyświetlenie punktu }
  PrzesunDo(400, 200);     { Przesunięcie punktu }
  Ukryj;                   { Ukrycie punktu }
end;
```

Należy zwrócić uwagę na rozróżnianie pojęć **typ obiektowy** i **obiekt**, ponieważ są one często mylone. Typ obiektowy, zwany też **klasą**, określa sposób interpretowania zawartości obszaru pamięci poprzez ustalenie charakteru zawartych w nim informacji i dostarczenia metod ich przetwarzania. Natomiast obiekt jest reprezentantem typu obiektowego zajmującym obszar pamięci, zmienną typu obiektowego. Zależność pomiędzy typem obiektowym a obiektem widać na przytoczonym wyżej przykładzie: `TPunkt` jest typem obiektowym, a *pkt* obiektem typu `TPunkt`.

Zaprezentowana wersja typu obiektowego `TPunkt` ma kilka wad. Po pierwsze, łatwy dostęp do pól obiektu może doprowadzić do niewłaściwego zachowania programu. Na przykład, po zmianie wartości pola *x* lub *y* obiektu *pkt* za pomocą instrukcji przypisania metoda `Ukryj` nie spowoduje ukrycia wyświetlonego punktu, gdyż informacja o jego właściwym położeniu została zagubiona. Po drugie, typ `TPunkt` nie jest dostosowany do roli typu bazowego. Naturalnym oczekiwaniem jest, by na jego podstawie można było tworzyć typy pochodne reprezentujące inne figury geometryczne (okrąg, prostokąt itp.), które różniłyby się jedynie wyglądem, a pozostałe cechy dziedziczyły (wyświetlanie, przesuwanie, ukrywanie). Nie umożliwia tego metoda `Rysuj` kreśląca punkt na ekranie. Nie jest ona wirtualna, nie można więc zmienić jej działania i wykorzystać do rysowania innych figur w obiektach potomnych. Zdefiniowanie jej jako wirtualnej pozwoli na jej redefinicję w dowolnym typie pochodnym. Po trzecie wreszcie, wraz z typem `TPunkt` należy zdefiniować związany z nim typ wskaźnikowy, który umożliwi dynamiczne tworzenie obiektów i list obiektów.

Dostęp do składowych obiektu

W Turbo Pascalu rozróżnia się dwa poziomy dostępu do pól i metod obiektu, określane słowami kluczowymi dzielącymi definicję typu obiektowego na sekcje[2]:

- `private` (prywatne) — dostępne tylko w module definiującym typ obiektowy,
- `public` (publiczne) — dostępne z dowolnego miejsca programu.

Jeżeli nie określono poziomu dostępu, składowe obiektu są publiczne. Zaleca się, aby pola były deklarowane jako prywatne, metody jako publiczne, a pobieranie wartości pól i ich aktualizacja odbywały się za pomocą metod. Pozwala to na precyzyjne kontrolowanie dostępności do danych obiektu.

Metody wirtualne, konstruktor i destruktor

Metodę określa się jako wirtualną, umieszczając po jej nagłówku w definicji typu słowo kluczowe `virtual`. Jeżeli nie ma pewności, że metoda nie będzie redefiniowana w typie pochodnym, lepiej zdefiniować ją jako wirtualną. W ten sposób uniknie się kłopotliwego modyfikowania kodu, gdyby okazało się, że w nowym typie potomnym jej działanie ma być odmienne i trzeba utworzyć jej nową wersję. Jeżeli typ obiektowy zawiera metodę wirtualną, nazywany jest **polimorficznym**.

Każdy typ polimorficzny musi posiadać specjalną metodę, zwaną **konstruktorem**, definiowaną z użyciem słowa kluczowego `constructor` zamiast `procedure`. Zadaniem konstruktora jest inicjalizacja obiektu. Oprócz przewidzianego w kodzie nadawania wartości początkowych polom obiektu, konstruktor wykonuje pewne czynności organizacyjne związane z obsługą metod wirtualnych. Każdy obiekt typu polimorficznego powinien być zainicjalizowany poprzez wywołanie konstruktora, przy czym powinno to nastąpić zanim wywołana zostanie jakakolwiek jego metoda wirtualna. Konsekwencje

[2] W Object Pascalu, podobnie jak w C++, jest jeszcze poziom `protected` (chronione).

niezastosowania się do tego wymagania mogą być różne; może nastąpić nawet załamanie systemu.

Typy polimorficzne często zawierają inną metodę specjalną, zwaną **destruktorem**, definiowaną za pomocą słowa `destructor` zamiast `procedure`. Destruktor wykonuje pewne czynności czyszczące związane z likwidacją obiektu, np. zwolnienie pamięci przydzielonej obiektowi dynamicznie lub usunięcie z ekranu figury, którą reprezentuje obiekt. Zazwyczaj destruktor jest metodą wirtualną, co umożliwia modyfikację jego działania w obiektach potomnych.

Moduł Punkty w Pascalu

Najwygodniej jest zamknąć definicję typu obiektowego i jego metody w module. Pozwala to na wykorzystanie go w wielu innych programach. W sekcji interfejsu umieszcza się wówczas definicję typu, a w sekcji implementacji definicje metod. Zaletą takiego rozwiązania jest ukrycie szczegółowego kodu, którego modyfikacja nie jest potrzebna. Na wydruku 6.1 przedstawiona jest opracowana w ten sposób poprawiona wersja typu obiektowego `TPunkt`.

Wydruk 6.1. *Moduł Punkty.pas definiujący typ obiektowy reprezentujący punkt*

```
unit Punkty;

interface

type
  PPunkt = ^TPunkt;
  TPunkt = object
    constructor Init(xPoc, yPoc: integer);
    destructor Done; virtual;
    procedure Pokaz; virtual;
    procedure Ukryj; virtual;
    procedure Rysuj; virtual;
    function PodajX: integer;
    function PodajY: integer;
    function Widoczny: Boolean;
    procedure PrzesunDo(xNowe, yNowe: integer); virtual;
    procedure PrzesunO(xDelta, yDelta: integer); virtual;
  private
    x, y: integer;
    Wid : Boolean;
  end;

implementation

uses Graph;

constructor TPunkt.Init(xPoc, yPoc: integer);
begin
  x := xPoc;
  y := yPoc;
  Wid := false;
end;
destructor TPunkt.Done;
```

```
begin
  if Wid then Ukryj;
end;

procedure TPunkt.Pokaz;
begin
  Wid := true;
  Rysuj;
end;

procedure TPunkt.Ukryj;
var
  Color: word;
begin
  Color := GetColor;
  SetColor(GetBkColor);
  Wid := false;
  Rysuj;
  SetColor(Color);
end;

procedure TPunkt.Rysuj;
begin
  PutPixel(x, y, GetColor);
end;

function TPunkt.PodajX: integer;
begin
  PodajX := x;
end;

function TPunkt.PodajY: integer;
begin
  PodajY := y;
end;

function TPunkt.Widoczny: Boolean;
begin
  Widoczny := Wid;
end;

procedure TPunkt.PrzesunDo(xNowe, yNowe: integer);
var
  BylWidoczny: Boolean;
begin
  BylWidoczny := Wid;
  if BylWidoczny then Ukryj;
  x := xNowe;
  y := yNowe;
  if BylWidoczny then Pokaz;
end;

procedure TPunkt.PrzesunO(xDelta, yDelta: integer);
begin
  PrzesunDo(x+xDelta, y+yDelta);
end;

end.
```

Definicja typu `TPunkt` składa się z domyślnej sekcji publicznej obejmującej metody i sekcji prywatnej opisującej pola. Dostęp do pól poza modułem jest możliwy tylko za pomocą metod. Wartości pól *x*, *y* i *Wid* udostępniają proste metody `PodajX`, `PodajY` i `Widoczny`. Metody `PrzesunDo` i `PrzesunO` przesuwają punkt reprezentowany przez obiekt — pierwsza do pozycji o określonych współrzędnych, druga do pozycji przesuniętej względem jego położenia bieżącego o określony przyrost poziomy i pionowy. Konstruktor `Init` inicjalizuje obiekt, nadając wartości początkowe wszystkim jego polom. Destruktor `Done` usuwa punkt z ekranu, gdy obiekt jest niszczony[3]. Metody `Done`, `Rysuj`, `PrzesunDo` i `PrzesunO` są wirtualne; można więc będzie je przedefiniować w typach pochodnych, gdy zajdzie taka potrzeba.

Klasa w C++ i jej moduł definiujący

Kluczową rolę w programowaniu obiektowym w języku C++ pełni klasa, stanowiąca odpowiednik typu obiektowego w Turbo Pascalu. Definicja klasy przypomina definicję struktury. Składowymi klasy są pola i funkcje, a poziom dostępu do nich określają specyfikatory dostępu (słowa kluczowe zakończone dwukropkiem):

- `private` (prywatne) — dostępne tylko w klasie, w której są zdefiniowane,
- `protected` (chronione) — dostępne również w klasach pochodnych,
- `public` (publiczne) — dostępne w całym programie.

Brak specyfikatora dostępu oznacza, że składowe klasy są prywatne. Jak widać, mechanizm określania dostępu jest w C++ bardziej precyzyjny niż w Turbo Pascalu.

Nazwa konstruktora w C++ jest zawsze taka sama jak nazwa klasy, a nazwa destruktora również, z tym że jest poprzedzana znakiem tyldy (~). Obie funkcje nie zwracają żadnej wartości, nawet typu `void`, dlatego nie określa się ich typu.

Deklaracje funkcji wirtualnych poprzedza się w definicji klasy słowem kluczowym `virtual`. Funkcje udostępniające wartości pól są bardzo proste, toteż zazwyczaj definiuje się je, poprzedzając ich deklaracje słowem kluczowym `inline` lub umieszczając po nich właściwy kod funkcji [6, 14]. W miejscu wywołania takiej funkcji kompilator nie generuje żadnego kodu wywołującego, lecz udostępnia bezpośrednio wartość pola, co przyśpiesza wykonanie programu.

Najbardziej wygodnym sposobem przygotowania klasy do wykorzystania i udostępniania jest umieszczenie jej w module składającym się z pliku nagłówkowego, zawierającego definicję klasy, i pliku kodu źródłowego jej funkcji składowych. Moduł taki można skompilować i w takiej postaci go udostępniać. W celu uniknięcia wielokrotnego wstawienia definicji klasy do programu, gdy kilka wchodzących w jego skład modułów z niej korzysta, należy zastosować **kompilację warunkową** kodu źródłowego:

[3] Tradycyjnie w Turbo Pascalu konstruktorowi nadaje się nazwę `Init`, a destruktorowi `Done`, chociaż można te metody nazwać inaczej. W Object Pascalu zwykle nadaje im się nazwy `Create` i `Destroy`.

```
#ifndef Nazwa symboliczna
#define Nazwa symboliczna
   definicja klasy
#endif
```

Użyta nazwa symboliczna powinna być niepowtarzalna. Preprocesor sprawdza, czy nie jest ona zdefiniowana (dyrektywa `#ifndef`). Jeżeli nie jest, definiuje ją (dyrektywa `#define`) i wstawia tekst definicji klasy do programu, a jeżeli jest, pomija cały tekst aż do dyrektywy `#endif`. Tak więc definicja klasy zostanie wstawiona do programu tylko raz, bez względu na to, ile dyrektyw `#include` poleca wstawienie zawierającego ją pliku nagłówkowego.

Moduł Punkty w C++

Wydruk 6.2 przedstawia zbudowany według tego schematu plik nagłówkowy zawierający definicję klasy `Punkt` w języku Borland C++.

Wydruk 6.2. *Plik nagłówkowy Punkty.h modułu definiującego klasę reprezentującą punkt*

```
#ifndef  H_PUNKTY
#define  H_PUNKTY

class Punkt
{
   int x, y, wid;
public:
   Punkt(int xPoc, int yPoc);
   virtual ~Punkt();
   virtual void pokaz();
   virtual void ukryj();
   virtual void rysuj();
   int podaj_x()    { return x; };
   int podaj_y()    { return y; };
   int widoczny()   { return wid; };
   virtual void przesun_do(int xNowe, int yNowe);
   virtual void przesun_o(int xDelta, int yDelta);
};

#endif
```

W definicjach funkcji składowych klasy, występujących na zewnątrz definicji klasy, nazwy tych funkcji poprzedza się nazwą klasy i **operatorem zakresu** `::` (dwa dwukropki bez odstępu między nimi), który określa przynależność funkcji do danej klasy. Wydruk 6.3 przedstawia pełny kod źródłowy funkcji składowych klasy `Punkt` zdefiniowanej w powyższym pliku nagłówkowym.

Wydruk 6.3. *Plik Punkty.cpp kodu modułu definiującego klasę reprezentującą punkt*

```
#include <graphics.h>
#include "punkty.h"

Punkt::Punkt(int xPoc, int yPoc)
```

```
{
   x = xPoc;
   y = yPoc;
   wid = 0;
}

Punkt::~Punkt()
{
   if (wid) ukryj();
}

void Punkt::pokaz()
{
   wid = 1;
   rysuj();
}

void Punkt::ukryj()
{
   int color = getcolor();
   setcolor(getbkcolor());
   wid = 0;
   rysuj();
   setcolor(color);
}

void Punkt::rysuj()
{
   putpixel(x, y, getcolor());
}

void Punkt::przesun_do(int xNowe, int yNowe)
{
   int byl_widoczny = wid;
   if (byl_widoczny) ukryj();
   x = xNowe;
   y = yNowe;
   if (byl_widoczny) pokaz();
}

void Punkt::przesun_o(int xDelta, int yDelta)
{
   przesun_do(x+xDelta, y+yDelta);
}
```

Konstruktor `Punkt` inicjalizuje pola obiektu za pomocą instrukcji przypisania. Inne rozwiązanie polega na wykorzystaniu tzw. **listy inicjalizującej**, którą umieszcza się w definicji konstruktora po nawiasie zamykającym listę parametrów, poprzedzając dwukropkiem. Jej elementami są nazwy pól z zamkniętymi w nawiasach wartościami początkowymi. Konstruktor `Punkt` może więc być zdefiniowany inaczej:

```
Punkt::Punkt(int xPoc, int yPoc) : x(xPoc), y(yPoc), wid(0)
{}
```

A oto odpowiednik podanego na początku tego podrozdziału przykładu w Turbo Pascalu ukazujący wykorzystanie klasy `Punkt`:

```
Punkt pkt(320, 240);          // Utworzenie niewidocznego punktu
pkt.pokaz();                  // Wyświetlenie punktu
pkt.przesun_do(400, 200);     // Przesunięcie punktu
pkt.ukryj();                  // Ukrycie punktu
```

Najpierw tworzymy i pokazujemy nowy punkt pośrodku ekranu, następnie przesuwamy go w nowe miejsce i na koniec ukrywamy. Wyszczególnione w deklaracji zmiennej *pkt* parametry są potrzebne do zainicjalizowania obiektu przez konstruktor.

Moduł Ruch przesuwania punktu w Pascalu

Naszym zamierzeniem jest teraz napisanie programu, który umożliwiałby przesuwanie punktu na ekranie za pomocą klawiszy strzałek: góra, dół, lewo, prawo. Operacja przesuwania punktu po ekranie, sterowana klawiaturą, wystąpi w kilku przytoczonych niżej programach, najlepiej więc wyrazić ją w postaci podprogramu i umieścić w module. Podprogram ma pobierać znaki z klawiatury i reagować na klawisze strzałek przesunięciem punktu, zgodnie z ich oznaczeniami. W przypadku napotkania znaku *Esc* podprogram ma zakończyć działanie i zignorować pozostałe znaki. Jest oczywiste, że będzie pobierał znaki z klawiatury za pomocą wygodnej funkcji zdefiniowanej w module `Klaw`, omówionym w podrozdziale „Kalendarz miesięczny” w rozdziale 2.

Wydruk 6.4 przedstawia moduł przesuwania punktu w Turbo Pascalu. Pierwszym parametrem procedury `Przesun` jest zmienna obiektowa reprezentująca przesuwany punkt. Pozostałe dwa parametry określają wielkość przesunięcia jednostkowego wzdłuż osi x i y. Formalnie nieskończona pętla `repeat...until` zostaje przerwana za pomocą procedury `Break` w momencie wczytania znaku *Esc*.

Wydruk 6.4. *Moduł Ruch.pas zawierający procedurę przesuwania punktu*

```
unit Ruch;

interface

uses Punkty, Klaw;

procedure Przesun(var p: TPunkt; dx, dy: integer);

implementation

procedure Przesun(var p: TPunkt; dx, dy: integer);
begin
  repeat
    case CzytajZnak of
      k_UP   : p.PrzesunO(0, -dy);
      k_DN   : p.PrzesunO(0, dy);
      k_LEFT : p.PrzesunO(-dx, 0);
      k_RIGHT: p.PrzesunO(dx, 0);
      k_ESC  : Break;
    end {case};
  until false;
end;

end.
```