6

Elementy Pascala

6.1. Konstrukcja programu w Pascalu

Poniższy szablon stanowi uogólniony model. Jak nietrudno zorientować się po przejrzeniu przykładowych listingów zamieszczonych w książce, nie wszystkie elementy tego szablonu muszą występować w każdym przykładowym programie w języku Pascal. Szablon nie uwzględnia również zaawansowanych stylów programowania strukturalnego, obiektowego czy zdarzeniowego. To jedynie prosty szablon elementarny dla programu sekwencyjnego, który nie został poddany żadnej formie modularyzacji.

```
Program etykieta; { nazwa programu (opcjonalna) }
Uses ... { lista używanych modułów }
Const ... { deklaracje stałych }
Type ... { deklaracje typów }
Var ... { deklaracje zmiennych }

BEGIN
... { główny blok programu }
END.
```

DEFINICJA

Identyfikator — każdy ciąg znaków złożony z symboli będących literą, cyfrą lub znakiem podkreślenia, rozpoczynający się literą lub podkreśleniem.

6.2. Typy danych

Podstawowe typy danych mają decydujący wpływ na tworzenie deklaracji zmiennych prostych i agregatów danych. Znajomość typów danych jest niezbędna również do poprawnego tworzenia deklaracji i definicji funkcji oraz procedur. O zakresie stosowalności decydują zazwyczaj dwa podstawowe parametry — zakres dostępnych war-

tości i wielkość zajmowanego pola pamięci (w bajtach). Warto zwrócić uwagę, że od wyboru typów danych zależy w sposób bezpośredni wielkość pliku wykonywalnego (zmienia się ilość zarezerwowanej pamięci), a w sposób pośredni — prędkość działania kodu. Do zademonstrowania różnic w czasach wykonania operacji w zależności od typu danych można wykorzystać przykładowe programy ilustrujące pomiar czasu rzeczywistego. W Pascalu do pomiaru czasu rzeczywistego z dokładnością do 1/100 sekundy można wykorzystać procedurę biblioteczną `GetTime()`.

6.2.1. Typy proste i porządkowe

Typ wyliczeniowy

```
Type identyfikator = (lista_identyfikatorów);


{ Przykład: }


Type DniTygodnia = (poniedzialek, wtorek, sroda, czwartek, piatek, sobota,
niedziela);
```

Analogią w C/C++ jest typ wyliczeniowy (patrz: enum).

Typy numeryczne całkowite

Nazwa	Zakres
Shortint	[–128, 127]
Byte	[0, 255]
Integer	[–32 768, 32 767]
Word	[0, 65 535]
Longint	[–2 147 483 648, 2 147 483 647]

Typ logiczny

```
Boolean = (False, True)
```

Typ znakowy char — 1 bajt

```
Var Znak : Char;
```

Typ okrojony

```
Type dwucyfrowe = 10 .. 99;

Type litera = 'A' .. 'Z';

Type dni_robocze = poniedzialek .. piatek;
```

Typy numeryczne rzeczywiste

Nazwa	Wielkość (patrz: SizeOf(...))
Real	6 bajtów
Single	4 bajty
Double	8 bajtów
Extended	10 bajtów

Dane tekstowe: typ łańcuchowy

```
Type imie = String[25];
```

6.2.2. Typy strukturalne

Tablice: typ tablicowy

```
Type wektor = Array[0..20] Of Integer;
Type macierz1 = Array[1..10] Of Array[1..20] Of Real;
Type macierz2 = Array[1..20,1..20] Of Real;
```

Struktury danych: typ rekordowy

```
Type
 Data = Record
  dzien : 1 .. 31;
  miesiac : 1 .. 12;
  rok : Integer;
 End;

Type
 DanePersonalne = Record
  nazwisko : string[25];
  imie : string[18];
  data_urodzenia : Data;
 End;
```

Zbiory: Set Of

```
Type dni_pracy = Set Of dni_robocze;
```

6.2.3. Definiowanie typów danych użytkownika/programisty

```
Type liczba = Integer;
```

6.3. Stałe i zmienne — deklarowanie i inicjowanie

W językach strukturalnych deklaracja zmiennej, funkcji lub procedury jest konieczna przed jej pierwszym użyciem w programie. O ile BASIC (język interpretowany) nie wymaga deklaracji zmiennej, o tyle Pascal czy C wymagają, by deklaracje zmiennych nastąpiły przed początkiem części operacyjnej, czyli instrukcji. C++ jest pod tym względem bardziej elastyczny, ponieważ pozwala deklarować zmienną bezpośrednio przed jej użyciem. Miejsce deklaracji zmiennej decyduje o zakresie jej dostępności (widoczności). Zmienne deklarowane na początku, przed częścią operacyjną programu, to zmienne globalne. Zmienne deklarowane w obrębie funkcji/procedur to zmienne lokalne. Zmienne globalne są zazwyczaj automatycznie zerowane, ale wartość zmiennych lokalnych przed ich zainicjowaniem w programie może pozostawać przypadkowa. Deklaracja zmiennej powoduje zarezerwowanie dla niej pamięci. Zainicjowanie zmiennej powoduje przypisanie jej początkowej wartości.

6.3.1. Deklaracje zmiennych

```
Var
 x : Integer;
 y : Real;
 b1, b2 : Boolean;
```

6.3.2. Deklarowanie i inicjowanie stałych i zmiennych

```
Const Pi : Real = 3.14;
znak : Char = 'A';
Const C : Real = 123;
Var X : Real;
....
X := 12345;
...

Var
 a : wektor;
```

```
 b : macierz;
 c : Array[1..10] Of wektor;
 d : Array[1..10] Of macierz;

a[2] := 13.456;
b[3][4] := 12.34;
b[2,3] := 11.12345;

{Konkatenacja, czyli laczenie ciagow znakow}
{Rezultat:                                  }
'To ' + 'Oddzielne'+' '+'teksty'  → 'To Oddzielne teksty'
```

6.4. Deklarowanie i inicjowanie stałych i zmiennych strukturalnych (operator kropki)

Zmienne strukturalne (podobnie jak zwykłe zmienne) mogą być zmiennymi globalnymi lub lokalnymi. Jeśli rekord (tablica rekordów) jest zmienną globalną, zazwyczaj zostanie automatycznie wyzerowany. Jeśli rekord jest zmienną lokalną, to do momentu zainicjowania zmienna taka może zawierać zupełnie przypadkową zawartość. Próba użycia niezainicjowanej zmiennej może powodować trudny do przewidzenia i zdiagnozowania błąd w działaniu programu. W zmiennych typu `String` w Pascalu pierwszy bajt określa liczbę znaków, dlatego automatyczne wyzerowanie powoduje uznanie tekstu za „łańcuch o zerowej długości”.

```
Type
 Data = Record
  dzien : 1..31;
  miesiac : 1..12;
  rok : Integer;
 End;

Var
 data_ur : Data;
 facet : Record
  nazwisko : string[25];
  imie : string[20];
  data_ur : data
 End;
```

```
Begin
 data_ur.dzien := 15;
 data_ur.miesiac := 5;
 data_ur.rok := 1988;
 facet.nazwisko := 'Burak';
 facet.imie := 'Jan-Maryja';
 facet.data_ur.rok := 1988;
 facet.data_ur := data_ur
 {...}
```

6.5. Operatory Pascala

`+`,`-` — zmiana znaku (operatory jednoargumentowe)

`@` — operator adresowy, pozwala zainicjować wskaźnik

`Not` — negacja logiczna

`*`, `/`, `Div`, `Mod`, — mnożenie, różne formy dzielenia

`And` — iloczyn logiczny

`Shl`, `Shr` — przesunięcie bitów w lewo/w prawo

`+`, `-` — dodawanie/odejmowanie arytmetyczne

`Or`, `Xor` — alternatywa zwykła i wyłączna

`=`, `<>`, `<`, `>`, `<=`, `>=` — operatory relacji

`In` — operator przynależności do zbioru

6.6. Wyrażenia

Wyrażenia w rozumieniu Pascala to:

- stała,
- zmienna,
- wywołanie funkcji,
- wyrażenia połączone operatorem dwuargumentowym.

UWAGA

Wobec wyrażeń można stosować operatory relacji (np. porównywać wyrażenia). Relacje dają w wyniku wartości typu `Boolean`, a działają także dla łańcuchów tekstowych (wg kodów ASCII znaków).

6.7. Operacje na zbiorach (teoria mnogości)

Operacje na zbiorach (*Set*) pozwalają na łatwe wyznaczenie części wspólnej i różnicy zbiorów. W najprostszym przypadku (jak w przykładzie poniżej) elementami zbioru są liczby całkowite.

+, -, * — suma, różnica i iloczyn zbiorów o zgodnych typach elementów. Przykład:

	rezultat:
[2,3,4]+[4,5]	[2,3,4,5]
[2,3,4]-[4,5,6]	[2,3]
[3,4]*[4,5,6]	[4]

6.8. Instrukcje Pascala

Instrukcje elementarne zakończone są średnikiem. Instrukcja złożona (blok instrukcji) ujmowana jest w tzw. nawiasy programowe, czyli parę słów kluczowych `Begin-End`. Jest jeden wyjątek: średnika nie stawia się przed słowem kluczowym `Else` w instrukcji warunkowej.

```
{blok instrukcji sklada sie z instrukcji elementarnych}


Begin
 Instrukcja_1;
 ...
 Instrukcja_n;
End;
```

6.8.1. Instrukcja warunkowa If-Then-Else (Else jest opcjonalne)

```
If wyrazenie Then instrukcja;


{ lub }


If wyrazenie Then instrukcja1
Else instrukcja2;


{ Maksimum sposrod 2 liczb x i y }
If x>y Then maksimum := x Else maksimum := y;
```

```
{ Zagniezdzone if-then-else-if ... }

If wyrażenie_1 Then instrukcja_1
Else If wyrażenie_2 Then instrukcja_2
Else If wyrażenie_3 Then instrukcja_3
Else { ... };
```

6.8.2. Instrukcja wyboru wielokrotnego Case

```
Case wyrazenie Of
 sekwencja_instrukcji_wyboru
End;

{ lub }

Case wyrazenie Of
 sekwencja_instrukcji_wyboru
Else instrukcja;
End;

Case miesiac Of
 1,3,5,7,8,10,12 : dni := 31;
 2 : dni := 28;
 4,6,9,11 : dni := 30;
End;

Case miesiac Of
 4,6,9,11 : dni := 30;
 2 : dni := 28;
 else dni := 31;
End;
```

6.8.3. Instrukcja For

```
For zmienna := wyr_1 To wyr_2 Do instrukcja;
{ lub }
For zmienna := wyr_1 DownTo wyr_2 Do instrukcja;
```

6.8.4. Instrukcje While i Repeat

```
While wyrażenie Do instrukcja;

Repeat instrukcja Until wyrażenie;
```

UWAGA

1. Aby skonstruować system funkcjonalnie pełny, niezbędne są dwie instrukcje pętli. Jedna musi sprawdzać warunek iteracji na wejściu (w Pascalu jest to `While`), a druga na wyjściu (w Pascalu — `Until`). Inne instrukcje pętli (np. `For-To`, `For-Down-To`) nie są niezbędne, ale są wygodne w stosowaniu.
2. `Break/Continue` w Pascalu są procedurami bibliotecznymi, a w `C/C++` są słowami kluczowymi języka, ale ich rola i działanie pozostają takie same.

7

Funkcje i procedury

7.1. Obsługa standardowego wejścia-wyjścia

```
Write(lista_argumentow);
WriteLn(lista_argumentow);
Read(lista_argumentow);
ReadLn(lista_argumentow);
```

Wersje z końcówką `Ln()` dodatkowo zapisują/czytają znaki końca wiersza (`CRLF`).

Formatowanie: każdy argument powinien mieć postać:

- *wyrażenie*
- *wyrażenie : długość*
- *wyrażenie : długość : liczba_pozycji_dziesiętnych*

7.2. Definicja procedury

```
Procedure nazwa_p ( lista_parametrow );
deklaracje_zmiennych, stalych, itp.
Begin
 instrukcje
End;
```

7.3. Definicja funkcji

```
Function nazwa_f ( lista_parametrow ) : typ_rezultatu;
deklaracje_zmiennych, stalych, itp.
Begin
 instrukcje
End;
```

8

Obsługa wejścia-wyjścia. Operacje plikowe

8.1. Opis zmiennej plikowej

```
Type Osoba = Record
 nazwisko : string[25];
 imie : string[20];
End;

TypPlikowy = File Of Osoba;  { plik o zdefiniowanym typie elementow}
TypPlikowy2 = File;    { plik binarny, niezdefiniowany typ }
TypPlikowy3 = Text;    { plik tekstowy }
```

8.2. Skojarzenie zmiennej z fizycznym plikiem dyskowym, otwarcie pliku

```
Var
 plik : TypPlikowy;
 ...
 Assign(plik, 'C:\OSOBY.PLK');

{ Otwieranie i zamykanie pliku }
{ Reset - aby otworzyc istniejacy plik }
Reset(zmienna_plikowa);
```

```
Reset(zmienna_plikowa, rozmiar_zapisu);

{ Rewrite - aby utworzyc nowy plik }
Rewrite(zmienna_plikowa);
Rewrite(zmienna_plikowa, rozmiar_zapisu);

{ Append - aby dopisywac do pliku tekstowego }
Append(zmienna_plikowa);

{ Close - aby zamknac plik }
Close(zmienna_plikowa);
```

8.3. Operacje na danych pliku dyskowego (odczyt, zapis)

```
{ Dla plikow tekstowych }
Write(zmienna_plikowa, lista_argumentow);
WriteLn(zmienna_plikowa, lista_argumentow);
Read(zmienna_plikowa, lista_zmiennych);
ReadLn(zmienna_plikowa, lista_zmiennych);

{ Dla plikow zdefiniowanych, zdefiniowany typ elementu }
Write(zmienna_plikowa, lista_zmiennych_wyjsciowych);
Read(zmienna_plikowa, lista_zmiennych_wejsciowych);

{ Dla plikow niezdefiniowanych, binarnych }
BlockWrite(zmienna_plikowa, bufor, licznik);
BlockWrite(zmienna_plikowa, bufor, licznik, wynik);
BlockRead(zmienna_plikowa, bufor, licznik);
BlockRead(zmienna_plikowa, bufor, licznik, wynik);

{ Czy to koniec pliku? End Of File}
Eof(zmienna_plikowa);

{ Czy to koniec wiersza? – End Of Line}
Eoln(zmienna_plikowa);
```

```
{ Aktualna pozycja w pliku }
FilePos(zmienna_plikowa);

{ Rozmiar pliku }
FileSize(zmienna_plikowa);

{ Przejdz do wskazanej pozycji w pliku }
Seek(zmienna_plikowa, pozycja);

{ Pobierz biezacy katalog dysku okreslonego przez
Zmienna liczba (0 - dysk biezacy, 1 - A:, 2 - B:, 3 - C: ...) }
GetDir(liczba, zmienna_lancuchowa);

{ Zmien biezacy katalog }
ChDir(wyrazenie_lancuchowe);

{ Utworz katalog }
MkDir(wyrazenie_lancuchowe);

{ Usun pusty katalog }
RmDir(wyrazenie_lancuchowe);

{ Usun plik (zamkniety) }
Erase(zmienna_plikowa);
```