1. Język PASCAL

Pascal jest językiem programowania stworzonym przez Niklausa Wirtha z ETH w Zurichu w latach siedemdziesią tych. Język ten ze względu na swoją przejrzystą postać, możliwość budowy struktur i wielu innych zalet, szczególnie dobrze nadaje się do nauki jako pierwszy język programowania.

1.1. Program

W zasadzie będziemy mowić o TURBO PASCALu, który jest bardziej rozbudowany w porównaniu ze standardem.

Struktura programu zapisanego w PASCALu jest następująca:

```
[ nagłówek ]

[ deklaracje ]

  blok instrukcji
```

Nawiasy kwadratowe oznaczają, że zawarte w nich treści nie muszą wystąpić.

Blok instrukcji rozpoczyna się słowem `begin` i kończy się słowem `end` oraz kropką `.` - podobnie jak zdanie, które zaczyna się dużą literą, a kończy kropką; zdanie rozkazujące złożone. Ponieważ brak instrukcji jest instrukcją pustą to najprostszym programem napisanym w TURBO PASCALu jest

```
begin
end.
```

Nie robi on nic.

1.1.1. Nazwy

W każdym języku programowania, podobnie jak w języku potocznym używamy nazw abstrakcyjnych lub konkretnych pojęć. w języku programowania są to zapisane słowa. Nazywać musimy stałe występujące w programie, zmienne, procedury, funkcje, czasami sam program.

Nazwy w języku PASCAL rozpoczynają się zawsze od litery. Wielkość liter nie jest ważna. Dwie nazwy `NaZwa1, nazwa1` są w PASCALu tą samą nazwą.

1.1.2. Deklaracje typów

Pojęcie typu. Typy wielkości w PASCALu można podzielić na trzy główne grupy: typy skalarne i typy strukturalne, albo złożone i wskazy (lub wskaźniki). Do typów skalarnych zaliczamy typy standardowe takie jak `integer, real, boolean, char` oraz typy definiowane przez użytkownika, które dzielimy na typy wyliczeniowe oraz podzakresy.

Jeśli chodzi o typy strukturalne to należą do nich tablice (array), rekordy (record), zbiory (set) i pliki (file).

Wskazy stanowią bardzo szczególny typ wielkości. Są to takie wielkości, które pozwalają lokalizować w pamięci komputera inne wielkości używane w programie często bez jawnego odwoływania się do tych wielkości. Są związane z adresowaniem pamięci komputera.

1.1.3. Instrukcja przypisania

Najczęściej stosowaną instrukcją jest prawdopodobnie instrukcja przypisania. Znakiem przypisania w PASCALu jest `:=`. Użycie instrukcji przypisania pokazuje następujący program.

```
const
   a = 100;
ver
   b: integer;
begin
   b:=a;  {   <--- instrukcja przypisania  }
   writeln(a:5, b:5)
end.
```

Tutaj, zmiennej `b` przypisano wartość stałej `a`, która jest równa 100. Wynikiem działania programu powinna być linia tekstu zawierająca dwie liczby całkowite 100 zapisane obok siebie. Każda z nich zajmuje 5 pól.

```
  100  100
```

1.1.4. Instrukcje z wyborem (selekcje)

W języku TURBO PASCAL, selekcje są zrealizowane za pomocą trzech konstrukcji:

```
if WARUNEK then INSTRUKCJA;

if WARUNEK then INSTRUKCJA else INSTRUKCJA;

case ZMIENNA of
  nazwa1: INSTRUKCJA1;
  nazwa2: INSTRUKCJA2;
  ...
  nazwa3: INSTRUKCJA3
  else    INSTRUKCJA
end;
```

Zadanie 1. Schematy selekcji 2

Narysuj schematy blokowe instrukcji selekcji języka TURBO PASCAL. [*Spis*]

1.1.5. Instrukcje wejścia-wyjścia

Warto na początku omówić dwie ważne pary instrukcji języka TURBO PASCAL dotyczące czytania danych i wypisywania danych (wyników). Są to instrukcje: `read( parametry ),readln( parametry )` oraz: `write(parametry), writeln(parametry)`. Parametrami tych instrukcji są zmienne i opisy formatów danych. Będziemy je omawiać w miarę potrzeb. PRZYKŁAD. Poniższy program wczytuje proste dane.

```
{
UWAGA.
w nawiasach klamrowych można umieszczać uwagi
dotyczące wszystkiego.  Nawiasy te powodują,
że nie należąone do programu
i sa pomijane przez translator.
}

var
  a: real;      { a jest typu rzeczywistego }
  b: integer;   { b jest typu całkowitego }
  c: char;      { c jest typu znakowego }
begin
  Readln(a);    { czytaj z konsoli wartość a }
  Readln(b);    { czytaj z konsoli ,,        b }
  Readln(c);    { ,,                         c }
```

```
  Writeln(a);   { wypisz zawartość zmiennej a}
  Writeln(b);   { ,, }
  Writeln(c);   { ,, }
end.
```

1.1.6. Instrukcje powtarzania

W TURBO PASCALu są trzy instrukcje powtarzania. Oto one

```
while WARUNEK do INSTRUKCJA

repeat INSTRUKCJA until WARUNEK

for ZMIENNA:=POCZATEK to|downto KONIEC do INSTRUKCJA
```

W instrukcji `for` `ZMIENNA` jest typu wyliczeniowego podobnie jak zmienne `POCZATEK` i `KONIEC`, które ustalają zakres zmienności zmiennej `ZMIENNA`. w przypadku gdy spełniona jest relacja `POCZATEK` < `KONIEC` używa się słowa `to`. w przypadku przeciwnym słowa `downto`.

1.1.7. Typy złożone i ich deklaracje

1.2. Abstrakcyjne typy danych (ATD)

Pojęcie *abstrakcyjny typ danych* odnosi się do najważniejszych złożonych typów danych (każdego języka programowania). ATD jest kolekcją obiektów danych wraz z regułami operacji na nich. W TURBO PASCALu lub C++ zastępuje się je obiektami, klasami. Będziemy rozważać takie kolekcje danych jak
— tablice (array)
— rekordy (record)
— zbiory (sets)
— łańcuchy (strings)
— stosy (stacks)
— kolejki (queues)
— listy połączone (lists)
— drzewa (trees)
— grafy (graphs)
Postaramy się krótko scharakteryzować te typy danych oraz podać sposoby ich deklarowania i używania. Wszystkie mają bardzo szerokie zastosowanie we współczesnych metodach programowania.

Prócz tego postaramy się pokazać typowe operacje wykonywane na każdym z wymienionych typów. Przy okazji omówimy kilka algorytmów, które dzięki użyciu odpowiednich typów danych pozwoliły znaleźć efektywne sposoby rozwiązania starych problemów. Są to różne metody sortowania i przeszukiwania. To dzięki nim możemy szybko przeszukiwać encyklopedie i słowniki komputerowe zawierające dziesiątki tysięcy haseł.

Będziemy to poznawać na przykładach.

1.2.1. Tablice

McCracken, Salmon.
Tablica jest kolekcją elementów identyfikowanych przez zbiór indeksów wraz z operacją `Wstaw`, która definiuje (częściowe) odwzorowanie zbioru indeksów na zbiór elementów oraz operacją `Pobierz`, która wykorzystuje to odwzorowanie do odczytania elementów wcześniej umieszczonych w tablicy. Indeksem jest tu n-tka liczb całkowitych, np. (1) lub (17,4) itd.

Operacja `Wstaw` to najczęściej operacja przypisania := lub operacja bezpośredniego wczytania z wejścia. Dla przykładu, `A[7,4,1] := 34.6` lub `Read(A[7, 4, 1])`. w ostatnim przypadku `wejście` zawiera 34.6. Obie operacje powodują odwzorowanie indeksu (7,4,1) w element 34.6.

Operację `Pobierz` powinny poprzedzać operacje utworzenia tablicy i operacja `Wstaw`. Operacja utworzenia tablicy wiąże się z problemem reprezentowania tablicy. Jak rezerwować przestrzeń pamięci komputera i jak wielka ma ona być dla danej tablicy.

Najczęstszy sposób rezerwacji pamięci polega na sekwencyjnej reprezentacji tablicy. Jego walorem jest prostota, ekonomiczność (przy całkowitym wypełnieniu tworzonej tablicy) i własność tzw. losowego dostępu do tablicy. Polega on na tym, że można odczytać lub zapisać dowolny element tablicy podając tylko jego adres. Minusem jest strata pamięci w przypadku niepełnej zajętości tablicy (wtedy można stosować listy połączone).

Proces tworzenia tablicy sekwencyjnej rozpoczyna się w chwili gdy kompilator zarezerwuje adres w pamięci komputera, od którego rozpoczyna się tablica, adres jej pierwszego elementu. Użytkownika interesuje w zasadzie jak znaleźć pozostałe elementy. Problem ten rozwiązuje znajomość tzw. funkcji przydziału pamięci (allocation function). Funkcje te zamieniają indeks tablicy na adres pamięci. Rezerwacji pamięci dokonuje się przez użycie deklaracji:

```
var
   x: array[0..99] of real;
```

Funkcją położenia jest w tym wypadku

$$\mathrm{Loc}(X[i]) = \mathrm{Loc}(X[0]) + i\,.$$

Jeśli np. $\mathrm{Lox}(X[0]) = 1200$ to elementy następne wskazywane przez $\mathrm{Loc}(X[i])$ będą mieć numery $1201, 1202, \ldots 1299$. Pod adresem 1245 znajdzie się wartość przypisane elementowi $X[45]$. Jeśli deklaracja tablicy zawiera inny niż zerowy indeks dolny, np L (od *low*) i górny U (od *up*), to funkcja alokacji jest wtedy $\mathrm{Loc}(X[i]) = \mathrm{Loc}(X[L] + i - L$.

W przypadku gdy mamy tablice wielowymiarowe, np. 2-wymiarowe o rozmiarze $[0..n] \times [0..m]$ elementów, funkcja alokacji jest

$$\mathrm{Loc}(X[i,j]) = \mathrm{Loc}(X[0,0] + i(n+1) + j\,.$$

Deklaracja takiej tablicy, nazwijmy ją $A(i,j)$, ma w PASCALu jedną z postaci:

```
const
   n=19; m=29;

{ 1 }
var
  A: array [0..n] of array [0..m] of real;

{ 2 }
var
  B: array [0..n,0..m] of real;

{lub 3 }
type
   duZa=array [0..n,0..m] of real;
var
   C: duZa;
```

W przypadku tablic trójkątnych, 3-diagonalnych lub innych, które pojawiają się w różnych problemach (diagonalizacja, układy równań liniowych itd.) rezerwowanie pamięci komputera w taki sam sposób jak poprzednio jest rozrzutnością. Jak to robić ekonomicznie? Należy inaczej lokować elementy tablic.

Jako przykład rozpatrzymy tablicę trójkątną.

$$T = \begin{Bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ \dots & & & \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_n \end{Bmatrix}$$

Odwzorujemy ją na tablicę liniową L. Mamy

$$t_{11} \to L_1 \quad T_{12} \to L_2, \quad T_{21} \to L_3, \text{ itd.}$$

Możemy w tym przypadku zapisać funkcję położenia w postaci

$$\mathrm{Loc}(X[w,k]) = \mathrm{Loc}(X[1,1]) + I\,,$$

gdzie I jest przesunięciem względem elementu pierwszego $X[1,1]$. Można się domyślać, że związek pomiędzy numerem wiersza w, kolumny k i wymiarami tablicy jest liniowy. Zapiszmy więc

$$I = wx + ky + z\,,$$

gdzie x, y, z są stałymi, które należy wyznaczyć. Z przyporządkowania pierwszych trzech elementów $T \to L$ mamy:

$$\begin{aligned} x + y + z &= 0\,, \\ x + 2y + z &= 1\,, \\ 2x + y + z &= 2\,. \end{aligned}$$

Stąd $x = 2$, $y = 1$, $z = -3$. Mamy wobec tego

$$\mathrm{Loc}(X[w,k]) = \mathrm{Loc}(X[1,1]) + 2w + k - 3\,,$$

itd.

Często nie opłaca się rezerwować pamięci komputera na dane, z którymi pracujemy. Przykładem może być macierz Hilberta, której elementami są liczby $1/(w + k - 1)$, gdzie k i w są numerami kolumny i wiersza. Lepiej jest w tym przypadku napisać funkcję, która je obliczy.

```
function Hilbert(k, w: integer): real;
begin Hilbert:=1.0/(w+k-1.0) end;
```

1.2.2. Rekordy

Rekord stanowi kolekcję pól identyfikowanych nazwą oraz operacje `Wstaw` i `Pobierz`. Każde pole może być dowolnego określonego standardowego lub zdefiniowanego wcześniej typu. Rekordy mogą być proste lub z wariantami.

PRZYKŁAD. Liczbę zespoloną z zapisujemy w postaci kartezjańskiej $z = x + iy$ albo w postaci biegunowej $z = re^{\Phi}$. W PASCALu możemy do tego celu użyć typu rekordowego.

```
type
   PostacZespolonej = (kartezjanska, biegunowa);
   Zespolona = record
                  case Postac : PostacZespolonej of
                     kartezjanska : (x, y : real);
                     biegunowa : (r, fi : real)
                  end;
```

Słowa **type, record, case, of, end** są słowami języka PASCAL. Jest to definicja typu zmiennych (`Zespolona`), którymi będziemy się posługiwać w programie. Słowo `Postac` w definicji rekordu nazwiemy selektorem, który w tym wypadku może przyjmować tylko dwie wartości : `kartezjanska` i `biegunowa`. Jeśli selektor przyjmie wartość `kartezjanska` to rekord będzie zawierał dwa pola rzeczywiste nazwane tutaj `x, y`. Jeśli wartością selektora jest `biegunowa` to rekord składa się z pól `r, fi`. W ogólności nie wymaga się by warianty rekordu miały taką samą strukturę; mogą być dowolne.

Dopóki nie podamy wartości selektora nie można użyć żadnego z wariantów. Efektywna zawartość rekordu zmienia się więc z jego (selektora) wartością. Jeżeli rekord jest jednowariantowy to selektor nie występuje.

Podamy teraz przykład procedury, która czyta liczby zespolone.[1] PRZYKŁAD.

```
procedure CzytajZespolona(var z: Zespolona);
var
   Kod: char;
begin
   Read(Kod);
   Write(Kod);   (* Echo *)
   case Kod of
   'k': begin
           z.postac:=kartezjanska;
           Read(z.x);
           Read(z.y)
        end;
   'b': begin
           z.postac:=biegunowa;
           Read(z.r);
           Read(z.fi)
        end;
   else
        Write('Zle dane')
   end
end;
```

Pojawiły się nowe słowa: **procedure, var, begin, case, of, end**, które są słowami zastrzeżonymi języka; nie można ich używać poza standardowym znaczeniem.

Korzystając z definicji zmiennych typu `Zespolona` można zbudować procedury, które wykonują wszystkie działanie, a więc dodawanie, mnożenie, dzielenie itp. na zmiennych tego typu. Jako przykład napiszemy podprogram dodawania zmiennych typu `Zespolona`.

```
procedure SumujZ(z1, z2: Zespolona; var SumaZ: Zespolona);
(*
      Dodawanie liczb zespolonych, które moga byc
      reprezentowane w~postaci kartezjanskiej lub
      biegunowej
*)
var
   x1, y1, x2, y2: real;
begin
   if z1.postac = kartezjanska then
     begin x1:=z1.x; y1:=z1.y end
   else
```

[1] Zakładam, że znane są zwykłe instrukcje czytania wielkości takich typów jak **real, integer, char.**

```
      begin
         x1:=z1.r * cos(z1.fi);
         y1:=z1.r * sin(z1.fi)
      end;

   if z2.postac = kartezjanska then
     begin x2:=z2.x; y2:=z2.y end
   else
     begin
        x2:=z2.r * cos(z2.fi);
        y2:=z2.r * sin(z2.fi)
     end;

   SumZ.postac:=kartezjanska;
   SumZ.x:=x1+x2;
   SumZ.y:=y1+y2

end; (* SumZ *)
```

W procedurze, która została tutaj pokazana, najpierw rozpoznawany jest wariant zapisu liczby, a następnie wykonywane jest zwykłe dodawanie liczb rzeczywistych i umieszczanie wyników w polach rekordu SumZ. SumZ to nazwa funkcji wykonującej dodawanie.

Zadanie 2. Rekordy

2

Zmodyfikować procedurę `SumujZ` tak, by wynik był zapisany w postaci `biegunowa` w przypadku, gdy obie liczby są podane w tej postaci, a w postaci kartezjańskiej w innych przypadkach. [*Spis*]

Zadanie 3. Liczby zespolone 1

1

Napisać procedurę mnożenia liczb zespolonych `MultZ`. [*Spis*]

Zadanie 4. Liczby zespolone 2

1

Napisać procedurę odejmowania liczb zespolonych, która korzysta z procedury `SumujZ` i procedury negacji liczby zespolonej `MinusZ`, która zmienia znak liczby zespolonej na przeciwny (też napisać). [*Spis*]

Zadanie 5. Zety

2

Napisać procedurę dzielenia liczb zespolonych, która używa procedury mnożenia `MultZ` oraz procedury odwracania liczb zespolonych `InvZ`. [*Spis*]

Zadanie 6. Zety z tablicami

2

Powtórz wszystkie powyższe ćwiczenia używając tablic. [*Spis*]

Zadanie 7. Ułamki proste

3

Korzystając z definicji ulamka prostego napisz zbiór procedur i funkcji, które dodają, mnożą, dzielą, odejmują, czytają i wypisują ułamki, zawsze w postaci q/p. Skorzystaj ze struktury `record` a. Umieść napisane procedury w bibliotece modułów (Unit) o nazwie `ulamki`. Ułamki można upraszczać korzystając z algorytmu Euklidesa. Funkcję `Euklid` dodaj do biblioteki modułów. [*Spis*]

1.3. Zbiory (sets)

Zbiory są typem danych znanym w Turbo Pascalu. Do ich deklaracji służy konstrukcja `set of TypDanych`:

```
var
   A: set of WyliczeniowyTypDanych;
```

Słowo `WyliczeniowyTypDanych` oznacza typ danych wyliczeniowy, tzn. typ takich wielkości, które są uporządkowane w jakiś sposób, wg. jekiejś cechy. Mogą to być liczby typu `integer`, znaki (typ `char`) i inne. Podamy kilka przykładów. PRZYKŁAD.

```
type
   Dzien=(Po,Wt,Sr,Cz,Pt,So,Ni);
   Znaki=set of char;
   Cyfry=set of 0..9;
   Dni=set of Dzien;
   Litery=set of 'A'..'Z';
const
   cyfry_parzyste: Cyfry=[0,2,4,6,8];
   samogloski: Litery=['A','E','I','O','U','Y'];
var
   D: Dni;
```

Powyższe przykłady pokazują również sposoby używania tzw. konstruktorów zbiorów: "[]". Za pomocą tych nawiasów można tworzyć zbiory w deklaracjach `const` (stałych).

Na zbiorach można wykonywać takie operacje jak: sumowanie zbiorów ($+$; odpowiednik teoriomnogościowy: $\bigcup$), iloczyn (przecięcie) zbiorów ($*$; odpowiednik teoriomnogościowy: $\bigcap$), i różnica zbiorów ($-$). Ich wynik jest określony dokładnie tak jak w zwykłej teorii zbiorów.

Operatory zawierania $<=$, $>=$ mają taki sam sens jak operatory $\subset$ i $\supset$. Dodatkowo, operator `in` pozwala sprawdzić czy dany element należy do zbioru (np. `if (k in D) write('k należy do D');`).

1.3.1. Łańcuchy (strings)

Podobnie jak zbiory, łańcuchy stanowią podstawowy, złożony typ danych w TURBO PASCALu. Łańcuch to ciąg znaków (typ `char`) o dynamicznie określonej długości. Maksymalna długość łańcucha wynosi (dla TURBO PASCALa) 255.
PRZYKŁAD.

```
const
   DlugoscLinii=80;
type
   Nazwisko=string[25];
   Imie=string[25];
   Linia=string[80];
```

Z łańcuchami wiążemy następujące operatory

$$+ \quad = \quad <> \quad < \quad > \quad <= \quad >=$$

Operator $+$ jest operatorem łączenia łańcuchów (konkatenacja). Pozostałe operatory są operatorami relacyjnymi. Z łańcuchami wiążemy też funkcję `Length` (długość), która podaje długość łańcucha.
PRZYKŁAD.

```
var
   Nazwisko: string[25];
begin
   Nazwisko:='Kowalski';
   writeln('Dzien dobry panie ', Nazwisko)
end.
```

Wynikiem programu jest linia tekstu: `Dzień dobry panie Kowalski`. Zarówno zmienna `Nazwisko` jak i ciąg `'Dzień dobry panie '` są w tym programie łańcuchami. Ich długości i zawartości są róże.

Dodatkowe funkcje i operacje na łańcuchach (`Str`, `StrCat`, `StrCopy`, itd. są dostępne jako podprogramy w jednej z wielu bibliotek procedur i funkcji TURBO PASCALa: `Strings`. PRZYKŁAD.

```
{StrCopy.PAS}

{Sample code for the StrCopy function.}


{ For Windows: }
{ uses Strings, WinCrt; }

uses Strings;

var
 S: array[0..15] of Char;
begin
 StrCopy(S, 'Turbo Pascal');
 Writeln(S);
end.
```

Powyższy przykład jest kopią przykładu z pomocnika pakietu TURBO PASCAL w wersji 7.