

10. OPERACJE NA TABLICACH

10.1. Zmienna tablicowa

Dotychczas w tym podręczniku stosowaliśmy zmienne *proste*, to jest takie, w których można przechować pojedynczą wartość określonego typu. Teraz zajmiemy się opisem operacji na zmiennych *strukturalnych*, które pod wspólną nazwą mogą przechowywać wiele wartości. Jedną z takich zmiennych, mającą wiele zastosowań praktycznych, jest *zmienna tablicowa*, którą w skrócie będziemy nazywać *tablicą*.

Przykładową tablicę o nazwie *T* pokazano na rysunku 10.1. Zmienna *T* zajmuje w pamięci sześć kolejno umieszczonych obszarów, zwanych *elementami* tablicy, każdy o rozmiarze jednego bajta. W każdym z elementów można przechować po jednej liczbie typu *Byte*, więc w całej tablicy mieści się ich sześć. Liczby te mogą na przykład mieć wartości pokazane na rysunku.

	T
T [0]	45
T [1]	0
T [2]	33
T [3]	255
T [4]	17
T [5]	4

Rys.10.1. Przykładowa tablica jednowymiarowa

Każdy z elementów tablicy jest oznaczony indeksem, umieszczonym w nawiasach kwadratowych. W podanym przykładzie indeksy są kolejnymi liczbami całkowitymi z zakresu 0..5. W Turbo Pascalu do indeksowania można wykorzystać wartości dowolnych typów porządkowych. Zakres indeksów liczbowych może leżeć całkowicie lub częściowo w strefie ujemnych wartości. Jako indeksy mogą być wykorzystane również znaki, na przykład dla tablicy sześćoelementowej mogłyby to być kolejne litery: 'a' .. 'f'.

Tablica zawsze zawiera elementy tego samego typu, ale ten typ może być różny: liczbowy, znakowy, łańcuchowy, rekordowy a nawet tablicowy.

10.2 Definiowanie typy tablicowego i deklarowanie zmiennych tablicowych

Zmienne tablicowe różnią się cechami strukturalnymi:

- *zakresem indeksów* (który jednoznacznie określa liczbę elementów tablicy)
- *typem elementów* tablicy.

Dlatego przed zadeklarowaniem tablicy musimy formalnie opisać jej strukturę, czyli podać definicję typu tablicowego. Ogólna postać tej definicji jest następująca:

```
type Nazwa_typu = array [Zakres_indeksów] of Typ_elementów;
```

Jak widać, definicja typu zaczyna się od słowa kluczowego *type*. Następnie piszemy wybraną przez nas nazwę typu. Po znaku równości piszemy słowo kluczowe *array*, a po nim, w nawiasach

kwadratowych, wybrany zakres indeksów, dobrany w zależności od wymaganej liczby elementów. Następnie, po drugim słowie kluczowym *of*, podaje się typ elementów tablicy. Całość zapisu kończy się średnikiem.

Dla tablicy z rysunku 10.1 przyjęto jako nazwę typu słowo *Tab*. Zakres indeksów to *0..5*, a typem elementów tablicy jest *Byte*. Przy tych założeniach, definicja typu wygląda, jak następuje:

```
type Tab=array [0..5] of Byte;
```

Bardziej praktycznym rozwiązaniem jest zdefiniowanie stałych, określających indeks początkowy i indeks końcowy tablicy. Wtedy definicja typu przyjmie postać:

```
const Jp=0; Jk=5;
```

```
type Tab = array [Jp..Jk] of Byte;
```

Ten sposób jest wygodny, ponieważ w czasie pisania programu często decydujemy się na zmianę zakresu indeksów tablicy. Wtedy wystarczy jedynie zmienić wartości stałych *Jp*, *Jk*, a pozostałe instrukcje zależne od wartości indeksów mogą pozostać niezmienione.

Definicja typu jest tylko *opisem struktury danych*. Podanie definicji typu nie pociąga za sobą fizycznego przydzielenia pamięci w obszarze zmiennych globalnych. Aby to nastąpiło, należy zadeklarować zmienną lub zmienne tablicowe, korzystając z wcześniej zdefiniowanego typu, co nie różni się od sposobu, w jaki deklarowano zmienne typów prostych:

```
var T1, T2:Tab;
```

Po tej deklaracji, w pamięci operacyjnej komputera, w obszarze zarezerwowanym dla zmiennych globalnych, zostaną utworzone dwie zmienne o nazwach *T1*, *T2* i wewnętrznej strukturze zgodnej z poprzednią definicją typu *Tab*. Każda z tych zmiennych jest zdolna do przechowania sześciu dowolnych wartości liczbowych typu *Byte*, więc zajmuje sześć bajtów pamięci.

Definiując typy tablicowe, trzeba pamiętać, że rozmiar struktury tablicy, opisany definicją typu, nie może przekroczyć 64 kilobajtów. Gdy przekroczymy tę wartość, pojawia się błąd kompilacji z komunikatem:

```
Error 22: Structure too large.
```

Może się zdarzyć, że rozmiar definiowanej struktury tablicowej nie przekracza dopuszczalnej wartości, ale deklarujemy kilka zmiennych tablicowych, które łącznie zajmują ponad 64 kilobajty pamięci. Wtedy zostaje przekroczony dopuszczalny rozmiar strefy, przeznaczonej w Turbo Pascalu dla zmiennych globalnych. Powstaje błąd kompilacji z komunikatem:

```
Error 96: Too many variables.
```

10.3. Zapewnianie tablicy wartościami początkowymi przy deklarowaniu.

Często, zwłaszcza przy uruchamianiu programów, wygodnie jest wypełnić tablicę określonymi wartościami początkowymi. Postępujemy podobnie, jak w przypadku inicjowania zmiennych prostych, używając słowa kluczowego *const* oraz znaku równości. Na przykład, dla tablicy z rysunku 10.1 napiszemy:

```
const T:Tab=(45, 0, 33, 255, 17, 4);
```

Jak widać, inicjowane wartości elementów tablicy pisze się kolejno w nawiasach zwykłych; wartości te oddziela się przecinkami. Liczba wpisanych wartości musi być zgodna z liczbą elementów tablicy, określoną w definicji typu. Pamiętajmy, że mimo użycia słowa *const*, tablica *T* jest *zmienną* i jej zawartość może ulegać zmianie w czasie pracy programu.

10.4. Odwoływanie się do elementów tablicy

Do dowolnego elementu tablicy można wpisać określoną wartość za pomocą instrukcji przypisania, lub odczytać ją z klawiatury. Wartość dowolnego elementu tablicy można wykorzystać w wyrażeniu, by wykonać obliczenia, można tę wartość wyprowadzić na ekran, skopiować do odpowiedniej zmiennej, zmodyfikować (zwiększyć, zmniejszyć, zmienić znak). Do tych wszystkich działań potrzebny jest sposób komunikowania się z danym elementem tablicy. (W programowaniu nazywamy to *odwoływaniem się* do elementu.) Odwołania są bardzo proste i polegają na wykorzystaniu indeksu, tak jak na rysunku 10.1. Chcąc na przykład zapisać wartość 88 do trzeciego elementu tablicy, który ma indeks 2, użyjemy instrukcji przypisania:

```
T[2]:=88;
```

albo wprowadzimy tę wartość z klawiatury wykonując instrukcję:

```
Readln(T[2]);
```

albo wreszcie skopiujemy ją z innej zmiennej, która przechowuje tę wartość:

```
A:=88;
```

```
T[2]:=A;
```

Trzeba pamiętać, że wartości indeksów mogą być wartościami zmiennych. Poniżej jako indeks służy bieżąca wartość zmiennej *J*:

```
J:=2;
```

```
T[J]:=88;
```

W miejscu indeksu można też wstawić odpowiedniego typu wyrażenie:

```
J:=5;
```

```
T[J div 2]:=88;
```

Znowu odwołaliśmy się tutaj do $T[2]$, ponieważ wyrażenie $J \text{ div } 2$ daje po obliczeniu wartość 2.

10.5. Zastosowanie instrukcji powtarzających do operacji na tablicach

Bardzo wiele operacji tablicowych polega na dokonywaniu tych samych działań na kolejnych elementach tablicy. W tego rodzaju operacjach jest oczywista przydatność instrukcji powtarzających – *while*, *repeat*, *for*. Instrukcja *for* jest szczególnie wygodna, ponieważ jej zmienna sterująca może być wykorzystana do indeksowania elementów tablicy. Wartości tej zmiennej zwiększają się o 1 po każdym cyklu pracy, a zakres tych zmian może być taki sam, jak zakres indeksów tablicy.

Na przykład, gdy tablica jest zdefiniowana jako:

```
type Tab = array[1..100] of Integer;  
var T:Tab;
```

to wszystkie jej elementy można wyzerować za pomocą instrukcji:

```
for J:=1 to 100 do T[J]:=0;
```

Wygodniejsze jest użycie dwóch stałych, określających początek i koniec zakresu indeksów:

```
const P=0; K:=100;  
type Tab=array[P..K] of Integer;  
var T:Tab;
```

Wtedy instrukcja zerowania elementów przyjmie postać:

```
for J:=P to K do T[J]:=0;
```

Przy tej postaci instrukcji, gdy zajdzie konieczność zmiany zakresu indeksów, nie trzeba będzie zmieniać w całym programie odpowiednich wartości we wszystkich instrukcjach *for* – wystarczy zmienić wartości stałych *P*, *K* na początku programu.

Jednak najbardziej wygodnym sposobem budowy instrukcji *for*, operującej na tablicy, jest zastosowanie standardowych funkcji *Low* i *High*. Argumentem tych funkcji może być nazwa zmiennej tablicowej lub nazwa typu tablicowego. Analizując definicję odpowiedniego typu tablicowego, funkcje *Low* i *High* zwracają odpowiednio dolną i górną wartość zakresu indeksów tego typu. Jeżeli zmienimy definicję typu tablicowego, to wszystkie instrukcje *for* obsługujące tablice automatycznie dostosują się do nowego zakresu indeksów. Instrukcja wykorzystująca funkcje *Low* i *High* wygląda, jak następuje:

```
for J:=Low(T) to High(T) do T[J]:=0;
```

W dalszych przykładach będziemy zawsze stosowali tę właśnie postać instrukcji *for*.

10.6. Ręczne i programowe wypełnianie tablicy

Przez ręczne wypełnienie tablicy będziemy rozumieli wprowadzenie do niej wartości z klawiatury. Odpowiednia procedura jest przedstawiona w przykładzie 10.1. Warto tutaj podkreślić, że każda procedura operująca na tablicy musi mieć argument tablicowy. W przeciwnym przypadku musiałaby pracować na tablicy globalnej, co jest niedopuszczalne z punktu widzenia zasad poprawnego programowania. Procedura wypełnia tablicę po to, by ją później, już wypełnioną, wykorzystywały inne procedury programu głównego. W takim razie argument tablicowy jest tutaj argumentem *wyjściowym*, który musi być poprzedzony słowem *var*. Oznacza to, że w istocie

Przykład 10_1. Procedury ręcznego i programowego wypełniania tablicy

```
{\$R+}
program Ex10_1;
uses Crt;
const P=0; K=9;
type Tab=array[P..K] of Word;
var Tablica:Tab;

procedure Zapklaw(var T:Tab);
  var J:Integer;
begin
  Writeln('Wprowadzaj elementy tablicy:');
  for J:=Low(T) to High(T) do begin
    Write('Element nr ',J,': ');
    Readln(T[J]);
  end;
end;

procedure Zaplos(var T:Tab; Z:Word);
  var J:Integer;
begin
  for J:=Low(T) to High(T) do T[J]:=Random(Z);
end;

{W programie głównym pokazano tylko instrukcje wywołania procedur.}
begin
  Zapklaw(Tablica);
  {-----}
  Zaplos(Tablica,100);
  {-----}
end.
```

procedura będzie operować na tej zmiennej tablicowej, której nazwy użyto w instrukcji wywołania w programie głównym. Gdyby potraktować argument tablicowy jako wejściowy (bez słowa *var*), to wypełniłaby się danymi lokalna tablica utworzona w obszarze stosu. Ponieważ, jak wiadomo, po zakończeniu procedury jej lokalne struktury są zwalniane, to wszystkie zapisane w tablicy dane zostałyby w tej sytuacji utracone.

Z uwagi na to, że tablice są na ogół dużymi strukturami, a domyślny obszar stosu niewielki (16 kilobajtów), często również wejściowe argumenty tablicowe poprzedza się słowem *var*. Dzięki temu nie kopiujemy tablicy z programu głównego do obszaru stosu, unikając ryzyka *przepiętienia stosu* (ang. *stack overflow*) przy wykonywaniu programu. Tak też będziemy postępować w dalszych przykładach – argument tablicowy zawsze będzie poprzedzony słowem *var*.

Na początku programu wprowadzono *dyrektywę kompilatora* $\{ \$R+ \}$. Zapis ten powoduje, że kompilator wykrywa, jako błąd, ewentualne przekroczenie zakresu indeksów. Jest to bardzo ważne, szczególnie przy wypełnianiu tablicy, ponieważ zapisanie danej do obszaru leżącego poza strefą, zarezerwowaną dla zmiennej tablicowej, może spowodować zniszczenie innej danej i pociągnąć za sobą nieprzewidziane konsekwencje.

Pierwsza z prezentowanych procedur pozwala za pomocą instrukcji *for* wprowadzić kolejno wartości wszystkich elementów tablicy. Użytkownik za każdym razem jest proszony o wprowadzenie elementu o podanym numerze. Podkreśmy, że za pomocą instrukcji *Readln* nie można od razu wczytać całej tablicy – instrukcja ta operuje jedynie na wybranych typach prostych.

Druga z procedur wypełnia tablicę programowo – bez udziału użytkownika. Za pomocą funkcji *Random* tablica wypełniana jest kolejno liczbami losowymi z przedziału 0..Z. Ponieważ Z jest argumentem procedury, za jego pomocą można dowolnie określać szerokość przedziału wpisywanych liczb losowych.

Obie instrukcje wywołania znajdujące się w programie głównym zawierają argument tablicowy *Tablica*. Nazwy *Tablica* dla zmiennej globalnej użyto specjalnie, by pokazać, że nazwa tablicy w programie głównym i nazwa tablicy w definicji procedury nie muszą, (choć mogą), być takie same. Argument określający zakres liczb losowych w wywołaniu procedury *Zaplos* podano jawnie jako 100. Oznacza to, że tablica wypełni się liczbami z zakresu 0..99.

10.7. Wyprowadzanie tablicy na ekran i jej sortowanie

W przykładzie 10.2 pokazano program, w którym zawartość zainicjowanej tablicy zostaje wyprowadzona na ekran, a następnie jej elementy zostają posortowane w porządku wzrastających wartości.

Za wyprowadzenie tablicy na ekran jest odpowiedzialna procedura *Poktab*, która może wyprowadzać wartości na dwa sposoby, w zależności od wartości argumentu *P*. Gdy *P=True*, elementy są pisane obok siebie w tym samym wierszu. Gdy *P=False*, elementy są wyprowadzane jeden pod drugim, w kolejnych wierszach ekranu.

Sortowanie jest wykonywane przez następną procedurę o nazwie *Bubblesort*. Przed sortowaniem tablica jest kopiowana do innej tablicy *T1*, by zachować jej poprzednią postać. Można przepisać tablicę w całości jedną instrukcją przypisania, ponieważ typ tablic *T* oraz *T1* jest ten sam. Zastosowano tak zwane *sortowanie bąbelkowe* (ang. *bubble sort*). Metoda polega na kolejnym sprawdzaniu relacji $T[K] > T[J]$, gdzie *J* i *K* są indeksami stworzonymi przez zmienne sterujące dwiema zagnieżdżonymi instrukcjami *for*. Po każdym cyklu zewnętrznej instrukcji *for*, kolejny element tablicy zajmuje właściwą pozycję w ciągu uporządkowanych wartości. Dla pokazania na ekranie etapów procesu sortowania, po każdym cyklu zewnętrznej pętli *for*, instrukcja *Poktab(T)* wyprowadza na ekran bieżącą postać częściowo posortowanej tablicy. Zauważmy, że wewnątrz jednej procedury własnej wywołujemy inną procedurę własną. Jest to oczywiście możliwe, pod warunkiem, że definicja procedury *Poktab* jest umieszczona w programie przed definicją procedury *Bubblesort*.

Po wykonaniu programu zobaczymy na ekranie następujący wydruk:

```
88 23 15 12 8 5 4 2
2 88 23 15 12 8 5 4
2 4 88 23 15 12 8 5
2 4 5 88 23 15 12 8
2 4 5 8 88 23 15 12
2 4 5 8 12 88 23 15
2 4 5 8 12 15 88 23
2 4 5 8 12 15 23 88
```

Przykład 10.2. Sortowanie tablicy i jej wyprowadzanie na ekran

```
{ $R+ }
program Ex10_2;
uses Crt;

const Jd=0; Jg=7;
type Tab = array [Jd .. Jg] of Byte;
const T:Tab=(88,23,15,12,8,5,4,2);
var T1:Tab;

procedure PokTab(var T:Tab; P:Boolean);
  var J:Integer;
begin
  for J:=Low(T) to High(T) do
    if P then Write(T[J]:4)
    else Writeln(T[J]:4);
  Writeln;
end;

procedure Bubblesort(var T,T1:Tab);
  var K,J:Integer;
  Kopia:Byte;
begin
  Poktab(T,True);
  T1:=T;
  for K:=Low(T) to High(T)-1 do begin
    for J:=K+1 to High(T) do
      if T[K]>T[J] then begin
        Kopia:=T[K];
        T[K]:=T[J];
        T[J]:=Kopia;
      end;
    Poktab(T,true);
  end;
end;

{program główny}
begin
  Clrscr;
  Bubblesort(T,T1);
  Readln;
end.
```

10.8. Przesuwanie cykliczne zawartości tablicy

W przykładzie 10.3 pokazano procedurę *Cyklprawo*, której zadaniem jest przesunięcie cykliczne zawartości tablicy o N pozycji w prawo. Aby zrozumieć działanie tej procedury, należy przedtem prześledzić trzy etapy procesu przesunięcia zawartości tablicy o *jedną* pozycję w prawo,

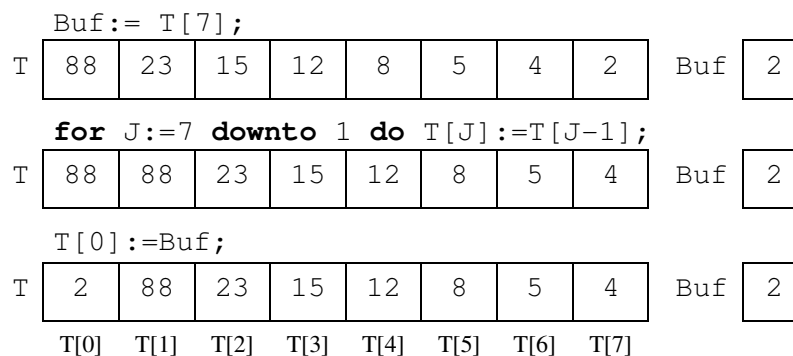
Przykład 10.3 Program z procedurą cyklicznego przesunięcia w prawo

```
{R+}
program Ex10_3;
uses Crt;
const Jd=0; Jg=7;
type Tab=array [Jd .. Jg] of Byte;
const T:Tab=(88,23,15,12,8,5,4,2);

procedure PokTab(var T:Tab; P:Boolean);
  var J:Integer;
begin
  for J:=Low(T) to High(T) do
    if P then Write(T[J]:4)
    else Writeln(T[J]:4);
  Writeln;
end;

procedure Cyklprawo(var T:Tab; N:Byte);
  var J,K,Buf:Integer;
begin
  for J:=1 to N do begin
    Buf:=T[High(T)];
    for K:=High(T) downto Low(T)+1 do T[K]:=T[K-1];
    T[Low(T)]:=Buf;
  end;
end;

begin
  Clrscr;
  PokTab(T,True);
  Cyklprawo(T,3);
  PokTab(T,True);
  Readln;
end.
```



Rys. 10.2. Cykliczne przesunięcie zawartości tablicy o jedną pozycję w prawo

w czym pomaga rysunek 10.2. Widać na nim, że pierwsza faza procesu polega na przepisaniu do bufora *Buf* ostatniej wartości w tablicy. Jest to konieczne, bo po przepisaniu wartości z pozycji 6 na pozycję 7, poprzednia wartość pozycji 7 zostaje wymazana. Druga faza procesu polega na przepisaniu kolejnych elementów $T[J-1]$ na miejsce ich sąsiadów po prawej stronie. Trzecia faza, to przepisanie z bufora do pierwszego elementu tablicy tej wartości, która początkowo zajmowała miejsce ostatnie.

Przedstawiony proces jest realizowany w procedurze *Cyklprawo* za pomocą trzech instrukcji umieszczonych w wewnętrznej instrukcji *for-downto*. Chcąc przesunąć zawartość tablicy o N pozycji, wystarczy N -krotnie powtórzyć ten proces. Służy do tego zewnętrzna instrukcja *for-to*. Mamy zatem w procedurze dwie zagnieżdżone pętle. Pamiętajmy, że każda z nich musi korzystać z odrębnej zmiennej sterującej.

10.9. Zliczanie elementów

Tak, jak to zrobiono w przykładzie 10.4, można sprawdzić, ile razy w tablicy wystąpiła wartość o określonych cechach. W tym celu organizuje się instrukcję *for*, która zawiera instrukcje warunkową *if*. Wyrażenie relacyjne umieszczone w tej instrukcji sprawdza, czy kolejny element ma interesujące nas cechy. Jeżeli test wypada pozytywnie, to instrukcja po słowie *then* zwiększa o 1 zmienną, służącą jako licznik wystąpień poszukiwanej wartości. Przed rozpoczęciem instrukcji *for* zmienną licznikową należy wstępnie wyzerować. W omawianym przykładzie pokazano definicję i wywołanie funkcji *Ileniep*, która sprawdza, ile jest w tablicy wartości nieparzystych. Warunek nieparzystości to $T[J] \bmod 2=1$, ponieważ liczba nieparzysta to taka, która daje resztę jeden w wyniku podzielenia jej przez dwa. Warto wiedzieć, że w Turbo Pascalu mamy do dyspozycji funkcję *Odd*, która zwraca wartość *True*, gdy jej argument jest nieparzysty. Mamy także standardową procedurę *Inc*, która po wywołaniu zwiększa swój argument o jeden. Zatem instrukcja sprawdzająca, użyta w funkcji *Ileniep*, mogłaby również wyglądać, jak następuje:

```
if Odd(T[J]) then Inc(Licznik);
```

Przykład 10.4. Definicja i wywołanie funkcji zliczającej nieparzyste elementy tablicy

```
{$R+}
program Ex10_4;
uses Crt;
const Jd=0; Jg=7;
type Tab=array [Jd .. Jg] of Byte;
const T:Tab=(88,23,15,12,8,5,4,2);
var Ile:Integer;

function Ileniep(var T:Tab):Integer;
var J,Licznik:Integer;
begin
  Licznik:=0;
  for J:=Low(T) to High(T) do
    if T[J] mod 2=1 then Licznik:=Licznik+1;
  Ileniep:=Licznik;
end;

{program główny}
begin
  Clrscr;
  Ile:=Ileniep(T);
  Write('Liczba nieparzystych elementow:',Ile);
  Readln;
end.
```

10.10. Szukanie wartości maksymalnej i jej miejsca w tablicy

W przykładzie 10.5 widzimy definicję funkcji *Szukmax*, której zadaniem jest znalezienie największej wartości pamiętanej w tablicy. Zgodnie z zasadami poprawnego programowania, pisanie wyników zostało powierzono odrębnej procedurze *Pokmax*.

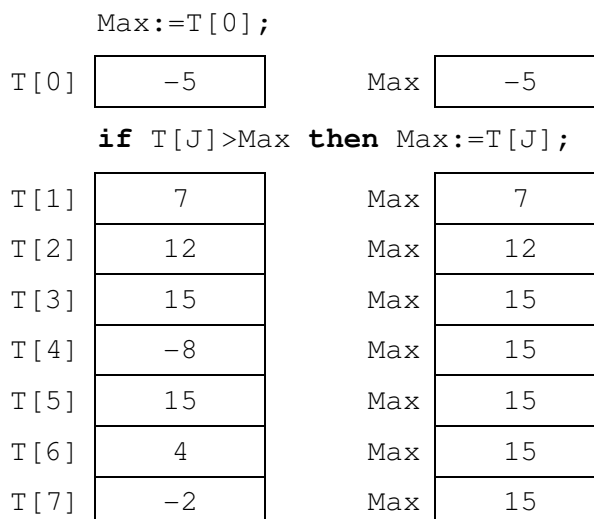
Przykład 10.5. Program znajdujący największą wartość w tablicy i jej położenie

```
{$R+}
program Ex10_5;
uses Crt;
const Jd=0; Jg=7;
type Tab=array [Jd .. Jg] of Integer;
const T:Tab=(-5,7,12,15,-8,15,4,-2);
var Max:Integer;

function Szukmax(var T:Tab):Integer;
  var J:Integer;
begin
  Max:=t[Low(T)];
  for J:=Low(T)+1 to High(T) do
    if T[J]>Max then Max:=T[J];
  SzukMax:=Max;
end;

procedure Pokmax(var T:Tab; Max:Integer);
  var J:Integer;
begin
  Writeln('Maksimum wynosi: ',Max);
  Write('i jest na pozycjach: ');
  for J:=Low(T) to High(T) do
    if T[J]=Max then Write(J:2,', ');
  Writeln(#8, '.');
end;

begin
  Clrscr;
  Max:=Szukmax(T);
  Pokmax(T,Max);
  Readln;
end.
```



Rys. 10.3. Ilustracja sposobu poszukiwania maksymalnego elementu tablicy

Prześledzimy sposób działania funkcji *Szukmax*, korzystając z rysunku 10.3. Do przechowywania bieżącej wartości maksimum służy zmienna *Max*. Na początku kopiujemy do niej początkowy element tablicy, następnie dla pozostałych elementów kolejno sprawdzamy warunek $T[J] > Max$. Jeżeli jest prawdziwy, wartość $T[J]$ zastąpi poprzednią wartość zmiennej *Max*. Dzięki takiemu postępowaniu, po zakończeniu instrukcji *for* w *Max* na pewno znajduje się największa z wartości przechowanych w tablicy. Zostaje ona przypisana nazwie funkcji.

Procedura *Pokmax* pobiera z programu głównego odszukaną wartość maksimum i wyprowadza na ekran odpowiedni komunikat. Ponadto, za pomocą swojej instrukcji *for* sprawdza wartości elementów, badając prawdziwość warunku $T[J] = Max$. Jeżeli warunek jest spełniony, to *J* czyli indeks elementu tablicy, w którym znaleziono *Max*, zostaje wyprowadzony na ekran. Po każdym indeksie wyprowadza się przecinek. Ciekawe jest przeznaczenie ostatniej instrukcji *Writeln*. Wyprowadza ona znak #8, czyli cofa kursor o jedną pozycję, by na miejscu ostatniego przecinka wydrukować kropkę, jako że wyprowadzony komunikat nie powinien kończyć się przecinkiem.

10.11. Tablice dwuwymiarowe

W praktycznych zastosowaniach bywają potrzebne tablice *dwuwymiarowe*, w których jednakowego typu elementy są ułożone w wierszach i kolumnach. Przykład takiej struktury pokazuje rysunek 10.4

		M[1,1]	M[1,2]	M[1,3]	M[1,4]	M[1,5]	M[1,6]
M[1]	1	2	3	4	5	6	
		M[2,1]	M[2,2]	M[2,3]	M[2,4]	M[2,5]	M[2,6]
M[2]	2	4	6	8	10	12	
		M[3,1]	M[3,3]	M[3,3]	M[3,4]	M[3,5]	M[3,6]
M[3]	3	6	9	12	15	18	

Rys. 10.4. Przykład tablicy dwuwymiarowej

Tablicę dwuwymiarową można interpretować jako jednowymiarową tablicę, której elementy są także jednowymiarowymi tablicami określonego typu. Na przykład tablica *M* na rysunku 10.4 jest trójelementową tablicą, której elementy są sześćelementowymi tablicami wartości typu *Byte*. Taką strukturę można w Turbo Pascalu zapisać na trzy sposoby:

1/ Jako jednowymiarową tablicę jednowymiarowych tablic:

```
type Typtab1 = array [1..3] of array [1..6] of Byte;
var M: Typtab1;
```

2/ Jako dwuwymiarową tablicę, przez podanie dwóch zakresów indeksów (indeksów wierszy oraz indeksów kolumn):

```
type Typtab2 = array[1..3,1..6] of Byte;
var M:Typtab2;
```

3/ W sposób pośredni – najpierw definiujemy typ tablicowy dla wiersza, a następnie używamy tego typu dla zdefiniowania całości struktury:

```
type Wiersz = array[1..6]of Byte;
Typtab3 = array[1..3]of Wiersz;
var M: Typtab3;
```

Trzeci sposób, pośredni, jest *zdecydowanie najlepszy*. Mając zdefiniowany typ dla wiersza, można łatwo dokonywać działań na całych wierszach. Na przykład zamiana miejscami wierszy $M[1]$ i $M[3]$ wygląda, jak następuje:

```
var Bufor: Wiersz;

  Bufor:=M[1];
  M[1]:=M[2];
  M[2]:=Bufor;
```

Oprócz odwoływania się do całych wierszy, dopuszczalne jest także kopiowanie tablicy do tablicy jedną instrukcją przypisania, pod warunkiem, że są zmiennymi tego samego typu:

```
var M, M1: Typtab3;

  M1:=M;
```

Do poszczególnych elementów tablic dwuwymiarowych odwołujemy się, podając najpierw indeks wiersza, a potem indeks kolumny. Równoważne są dwa sposoby zapisu:

```
M[J, K] :=0;
M[J][K] :=0;
```

Przykład 10.5 przedstawia operacje na tablicy dwuwymiarowej. Typowym narzędziem są w tym przypadku dwie zagnieżdżone instrukcje *for*, z dwiema różnymi zmiennymi sterującymi, które tu nazwano J , K . Wewnętrzna instrukcja *for* odwołuje się do kolejnych elementów wiersza, natomiast zewnętrzna instrukcja *for* powtarza proces dla kolejnych wierszy. Pokazano dwie proste procedury. Pierwsza, *Zaptab*, wypełnia tablicę w taki sposób, by były w niej pamiętane elementy tabliczki mnożenia liczb naturalnych z zakresu $1..16$. Druga procedura, *Poktab*, wyprowadza na ekran zawartość kolejnych wierszy tablicy.

Przykład 10.5. Program z tablicą dwuwymiarową

```
program Ex10_5;
uses Crt;
const N=16
type Row = array[1..N] of Word;
      Matrix = array[1..N] of Row;
var Macierz: Matrix;

procedure Zaptab(var M:Matrix);
var J,K:Integer;
begin
  for J:=1 to N do
    for K:=1 to N do M[J,K]:=J*K;
  end;
end;

procedure Poktab(var M:Matrix);
var J,K:Integer;
begin
  for J:=1 to N do begin
    for K:=1 to N do Write(M[J,K]:4);
    Writeln;
  end;
end;

begin
  Clrscr;
  Zaptab(Macierz);
  Poktab(Macierz);
  Readln;
end.
```

