Procedura HALT działa bez zmian, tzn. przerywa wykonywanie programu bez względu na miejsce, w jakim została wywołana. Procedura EXIT wywołana wewnątrz procedury lub funkcji utworzonej przez użytkownika powoduje zakończenie wykonywania jedynie danej procedury; wykonywanie programu jest kontynuowane od miejsca, w którym procedura ta została wywołana.

## 10.10. Tworzenie procedur i funkcji

System Turbo Pascal umożliwia tworzenie procedur i funkcji przez użytkownika. Do napisania programów realizujących algorytmy z rozdziału 8 tworzenie procedur nie jest niezbędne, w przypadku niektórych algorytmów jest jednak korzystne. Gdy osiągniesz pewną wprawę w ich tworzeniu, okaże się być może, że zyskałeś umiejętność mającą znaczenie nie tylko dla tworzenia programów w Turbo Pascalu, lecz także i dla Twojego sposobu myślenia o złożonych problemach z różnych dziedzin.

### 10.10.1. Pierwsza procedura

Fragment programu można wyodrębnić i nadać mu postać procedury. W programie fragment taki można wtedy zastąpić wywołaniem tej procedury. Jeżeli zastosować takie postępowanie do poszczególnych fragmentów programu realizujących poszczególne zadania cząstkowe, to program może się składać z wywołań kilku procedur. Wówczas staje się on przejrzystszy. Ponadto utworzone procedury można ewentualnie wykorzystywać w innych programach.

Procedura musi mieć nazwę. Definicja procedury zaczyna się od słowa PROCEDURE z nazwą procedury, po którym może wystąpić lista parametrów; dalsza część procedury ma taki sam układ jak program (deklaracje i definicje, a później część wykonawcza zawarta między słowami BEGIN i END), tyle że po końcowym słowie END stawia się średnik, a nie kropkę.

Zmienne zadeklarowane w programie głównym mogą być używane w procedurze bez ponownego deklarowania. Procedurę wywołuje się przez użycie w instrukcji jej nazwy (tak samo jak procedurę standardową). Definicje procedur umieszcza się po deklaracjach zmiennych, ale przed programem głównym.

Przypomnij sobie program TP-19. Występują w nim dwa identyczne fragmenty z pętlami FOR, służące do przedstawiania na ekranie zawartości całej tablicy *Porzadek*. Fragmenty te można zastąpić odwołaniem do procedury, tworząc jednocześnie odpowiednią procedurę. Uczynimy to w pro-

gramie TP-29; procedurze nadamy nazwę DRUKTABLPORZ. Porównaj oba programy.

## Program TP-29

```
program TP29;
  {tablice - przesuwanie elementow
   zmodyfikowany program TP20}
var
  i,j,ElementSkrajny: integer;
  Porzadek: array[1..6] of integer;

procedure DrukTablPorz;              {procedura drukowania}
begin
  for i:= 1 to 6 do Write(Porzadek[i]:10);
  WriteLn;
end; {DrukTablPorz}

begin  {program glowny}
  WriteLn('Wprowadz 6 liczb calkowitych (po kazdej nacisnij
                                                ENTER)');
  for i := 1 to 6 do ReadLn(Porzadek[i]);       {czytanie}
  DrukTablPorz;
  ElementSkrajny := Porzadek[1];
  for j := 1 to 5 do Porzadek[j] := Porzadek[j + 1];
                                              {przesuwanie}
  Porzadek[6] := ElementSkrajny;
  DrukTablPorz;
  Readln
end.
```

W programie głównym zamiast dwóch identycznych fragmentów są wywołania procedury DRUKTABLPORZ. Procedura jest zdefiniowana przed częścią wykonawczą programu. W nagłówku po słowie PROCEDURE jest jedynie jej nazwa. Pomiędzy BEGIN i END są instrukcje przeniesione z programu głównego. Za słowem END kończącym procedurę jest średnik (a nie kropka jak na końcu programu).

Komentarz ułatwia orientację, gdzie kończy się procedura. Stosowanie podobnych komentarzy jest przydatne w większych programach, zawierających większą liczbę definiowanych procedur (i funkcji).

### 10.10.2. Procedury bezparametrowe

Utworzona w programie TP-29 procedura DRUKTABLPORZ jest przykładem procedury bezparametrowej. Wykonuje ona zawsze takie same czynności na tych samych zmiennych. Samo istnienie takich procedur nie jest

chyba dla Ciebie zaskoczeniem; posługiwałeś się już na przykład procedurą CLRSCR.

Podstawową korzyścią ze stosowania procedur jest zwiększona przejrzystość programu. Mógłbyś w podobny sposób zamienić na procedurę każdy fragment programu realizujący pewne funkcje. Program główny mógłby po takich modyfikacjach sprowadzać się do wywołania procedur, np.

```
begin
  CzytanieDanych;
  DrukTablPorz;
  Przesuwanie;
  DrukTablPorz;
  Zakonczenie
end;
```

Przyznasz chyba, że jest bardziej czytelny niż program TP-19.

### Ćwiczenie 10-28

Zmodyfikuj program TP-29 tworząc procedury bezparametrowe czytania danych i ich przesuwania; także instrukcję READLN, powodującą zatrzymanie ekranu wynikowego, możesz przenieść do procedury (dodając komunikat polecający naciśnięcie klawisza). Uruchom program i sprawdź na przykładowych danych, czy działa prawidłowo.
W razie trudności porównaj tworzony przez siebie program z programem zapisanym w pliku TP29PBP.PAS.

Większa korzyść ze stosowania takich procedur występuje wtedy, gdy w programie występuje wiele identycznych fragmentów realizowanych przez procedury. Częściej jednak zdarzają się w programach fragmenty podobne, lecz nie identyczne. Wówczas przydaje się procedura o większym stopniu uniwersalności. Nie bez znaczenia jest też to, że raz napisaną procedurę możesz wykorzystywać w innych swoich programach. Uniwersalność procedury wiąże się ze sposobem wymiany informacji pomiędzy nią a programem głównym. Procedura DRUKTABLPORZ odczytuje i zapisuje zmienne zadeklarowane w programie głównym: elementy tablicy *Porzadek* tylko odczytuje, zaś zmienną *i* odczytuje i zapisuje. Nie mogłaby działać w odniesieniu do tablicy o innej nazwie; nie mogłaby działać również wtedy, kiedy w programie głównym brakowałoby deklaracji zmiennej *i*.

## 10.10.3. Zmienne lokalne i globalne

Zmienna *i* pełni funkcję pomocniczą i nie służy do przekazywania żadnych informacji między procedurą a programem głównym. Jeżeli cały program

zostaje przekształcony do ciągu wywołań procedur, to zmienna ta przestaje nawet być potrzebna w programie głównym.

W procedurze można stosować deklaracje i definicje w taki sam sposób jak w programie. Zmienne te „obowiązują" wówczas jedynie w obrębie procedury, tzn. jedynie tu można je odczytać i zapisać, natomiast w programie głównym, nie. Z tego powodu zmienne deklarowane w procedurze nazywa się **zmiennymi lokalnymi**. Zmienne deklarowane w programie głównym nazywa się **zmiennymi globalnymi**, ponieważ dostęp do nich jest zarówno w programie głównym, jak i w procedurach.

Zmienną pomocniczą *i* można zatem zadeklarować w procedurze, np.

```
procedure DrukTablPorz2;
var  i: integer;
begin
   for i:= 1 to 6 do Write(Porzadek[i]:10);
   WriteLn
end;
```

Zwiększa to uniwersalność procedury, ponieważ można ją wówczas stosować w programie, w którym nie ma deklaracji zmiennej całkowitej *i* albo wartość zadeklarowanej tam zmiennej *i* nie powinna być zmieniana.

Możesz zadać sobie pytanie, co się stanie, jeżeli w programie głównym zmienna o nazwie *i* również będzie zadeklarowana. Otóż nic groźnego. W procedurze jest „widziana" tylko ta zmienna *i*, która jest zadeklarowana w procedurze (lokalna). Przesłania ona zmienną *i* zadeklarowaną w programie głównym (globalną). Na przykład nadanie wartości zmiennej *i* w procedurze nie powoduje żadnej zmiany zmiennej globalnej *i*; podobnie zmiana wartości zmiennej globalnej *i* nie wpływa na wartość zmiennej lokalnej.

Zatem przyjmij następujące zalecenie, ułatwiające korzystanie z tworzonych procedur i funkcji.

> **Zapamiętaj!**
> Deklaruj w procedurze lub funkcji wszystkie używane w niej zmienne, które nie przenoszą informacji między nią a programem głównym.

## 10.10.4. Parametry procedur i funkcji

Zauważ, że zadeklarowanie w procedurze tablicy *Porzadek* zniweczyłoby sens jej działania. Tablica ta służy do przekazywania informacji między

różnymi fragmentami programu: tu są umieszczone liczby podane przez użytkownika, tu następuje ich przesuwanie i stąd muszą być pobierane do procedury drukowania zawartości tej tablicy. Procedura bezparametrowa odwołuje się bezpośrednio do tablicy zadeklarowanej w programie głównym, używając jej nazwy; zatem do tablicy o innej nazwie nie może być stosowana. Trzeba zatem do wymiany informacji posłużyć się **parametrami**.

Na przykładzie procedur i funkcji standardowych systemu Pascal mogłeś się przekonać o znaczeniu parametrów. Wywołując procedurę podawałeś z reguły parametr lub parametry, i wówczas działanie procedury odnosiło się do tych zmiennych i wyrażeń, które zapisałeś w instrukcji. Na przykład procedura WRITE drukowała tylko te napisy i wartości tych wyrażeń, które podawałeś jako jej parametry, a procedura INC zwiększała wartość wskazanej zmiennej. Podobnie było z funkcjami: ich wartość była obliczana na podstawie wartości tych wyrażeń, które podałeś jako ich argumenty (wyjątkiem były funkcje bezparametrowe, jak PI, i funkcje, które nie muszą mieć parametrów, jak RANDOM).

Zauważ też, że poszczególne parametry służą do wymiany informacji między programem i procedurą na jeden z dwóch sposobów:

- informacja jest przekazywana **w jednym kierunku**: wyłącznie z programu do procedury lub funkcji;
- informacja jest przekazywana **w obu kierunkach:** zarówno z programu do procedury, jak i z procedury do programu.

W pierwszym przypadku parametrami procedur i funkcji mogą być wyrażenia, ponieważ do procedury lub funkcji jest przekazywana tylko ich **wartość** (po ewentualnym obliczeniu), natomiast w drugim — wyłącznie **nazwy zmiennych**, ponieważ w procedurze może im zostać nadana wartość. Mówimy odpowiednio o przekazywaniu danych przez wartość lub przez zmienną (przez nazwę). Zauważ, że ten drugi przypadek dotyczy wyłącznie procedur. Do przekazywania informacji od funkcji do programu służy nazwa funkcji, a nie jej parametry.

Tak więc poprawne są instrukcje:

```
Inc(i);
WriteLn (i, j, i + j, napis, x, Sin(x));
j := 1 + Trunc(50 * Sin(x));
```

ale nie są poprawne instrukcje:

```
Inc(i + j);
ReadLn (i + j, Sin(x));
```

Nie nakazywałbyś przecież wykonania operacji przypisania o postaci

```
i + j := i + j + 1;
```

a temu byłaby równoważna pierwsza z wymienionych nieprawidłowych instrukcji (i podobna do niej byłaby częściowo druga).

W tworzonych przez siebie procedurach i funkcjach, parametrów używa się w analogicznym charakterze jak w procedurach i funkcjach standardowych.

Każda ze zmiennych użytych w Twojej procedurze lub funkcji musi być:

- albo zadeklarowana jako parametr (w nagłówku procedury),
- albo zadeklarowana jako zmienna lokalna (w treści procedury),
- albo nie zadeklarowana w procedurze (zmienna ta musi wówczas być zadeklarowana w programie zawierającym instrukcję wywołania procedury).

Deklarowanie zmiennych lokalnych już poznałeś; ma identyczną formę jak deklarowanie zmiennych w programie. (W podobny sposób możesz w procedurze lub funkcji definiować typy zmiennych, a nawet inne procedury i funkcje.)

Parametry deklaruje się w nagłówku procedury lub funkcji. Po nazwie procedury wymienia się je w nawiasie. Po każdym parametrze lub grupie parametrów jednakowego typu podaje się ten typ oddzielając go dwukropkiem od nazw parametrów, np.

```
procedure DrukujKomunikat (x: real; i: integer; lancuch:
                                                  string);
procedure DrukujSumy (i, j: integer; x, y, z: real);
```

Instrukcje wywołania tak zdefiniowanych procedur muszą zawierać oprócz ich nazw także podane w nawiasie wyrażenia odpowiadające poszczególnym parametrom (wartość wyrażeń musi mieć typ odpowiadający typowi parametru), np.

```
DrukujKomunikat(y + z, 25 DIV 2, 'warszawskie');
```

Parametry występujące w definicji procedury i funkcji są nazywane **parametrami formalnymi**, zaś parametry użyte w instrukcji wywołania procedury lub funkcji — **parametrami aktualnymi**.

W przypadku funkcji dodatkowo określa się typ wartości funkcji. Dopiero wtedy stawia się średnik, kończący nagłówek.

```
function NWDzielnik (i, j: integer):longint;
function Droga (v0, t, a: real):real;
```

Tak zdefiniowanych funkcji możemy w programie używać do nadawania wartości zmiennym (odpowiedniego typu), np.

```
NWWielokrotnosc := m * n / NWDzielnik (m, n);
```

Każdy parametr procedury lub grupę parametrów jednakowego typu można poprzedzić słowem VAR. Wówczas parametry te będą traktowane jako nazwy zmiennych, którym można nadawać wartość w treści procedury (i przekazywać w ten sposób wyniki do programu głównego), np.

```
procedure DodajElement (Element: real; var SumaElementow:
                                                    real);
```

Przy braku słowa VAR — jak w podanych poprzednio przykładach — system traktuje parametry z danej grupy jako służące jedynie do przekazywania wartości z programu głównego do procedury.

Jeżeli chcesz upewnić się, że rozumiesz różnicę działania procedur w przypadku zadeklarowania ich parametrów z użyciem słowa VAR bądź bez niego, to wykonaj poniższe ćwiczenie.

## Ćwiczenie 10-29

A. Przeanalizuj działanie programu TPVARPRO i określ, jakie liczby powinny być wydrukowane po wprowadzeniu zmiennej *Suma*.

```
program TPVARPRO;
var Elem, Suma: integer;
procedure DodajElement (Element: integer; SumaElementow:
                                                    integer);
begin
  SumaElementow := SumaElementow + Element;
  WriteLn ('druk w procedurze ',Element:10, SumaElementow:10)
end; {DodajElement}

begin
  ReadLn (Suma);
  ReadLn (Elem);
  WriteLn ('druk w programie  ',Element:10, SumaElementow:10);
  DodajElement(Elem, Suma);
  WriteLn ('druk w programie  ',Element:10, SumaElementow:10);
  ReadLn
end.
```

B. Przeprowadź taką samą analizę dla przypadku, gdy w definicji procedury DODAJELEMENT parametr *SumaElementow* będzie poprzedzony słowem VAR.
   Jak ma się przejawiać ewentualna różnica?
C. Odczytaj program z pliku TPVARPRO.PAS i przeprowadź eksperyment dla przypadków analizowanych w punktach A i B.
D. Zmień nazwy zmiennych deklarowanych w programie głównym: *Elem* na *Element* i *Suma* na *SumaElementow* (na takie same, jakie są użyte w definicji procedury DODAJELEMENT). Czy działanie programu powinno się zmienić? Jak?
E. Uruchom program ze zmienionymi nazwami zmiennych i sprawdź, czy miałeś rację.

Z używaniem tablic jako parametrów procedury wiąże się konieczność zdefiniowania ich typu w programie głównym (a nie tylko zadeklarowania). Na przykład chcąc w procedurze drukować elementy tablicy takiej jak *Porzadek*, nie można posłużyć się taką deklaracją jak w programie głównym:

```
procedure DrukWynikow (tablica:array[1..6] of integer);
```

lecz trzeba najpierw zdefiniować w programie głównym typ takich tablic:

```
type tabl6intyp = array[1..6] of integer;
```

i w nagłówku procedury powołać się na ten typ:

```
procedure DrukWynikow (tabl6intyp);
```

Tablica *Porzadek* musiałaby wówczas być zadeklarowana jako element typu *tabl6intyp*.

### Ćwiczenie 10-30

Spróbuj teraz przekształcić procedurę DRUKWYNIKOW na procedurę uniwersalną i odpowiednio zmodyfikuj program TP-29. Możesz porównać swój program z programem zapisanym w pliku TP29U.PAS.

## 10.10.5. Przykłady tworzenia procedur i funkcji

### Funkcja potęgowa

W systemie Turbo Pascal nie ma działania służącego do podnoszenia do potęgi (rozdz. 10.9.2). Tworzyłeś już program, który miał je realizować. Teraz utwórz procedurę (będziesz mógł posługiwać się nią w różnych programach).

```
function Potega(x,y:real):real;
begin
  potega := EXP(y * LN(x))
end;
```

## Przesuwanie cykliczne elementów tablicy

Powróćmy do programu TP-29 i przekształćmy jego fragment odpowiedzialny za przesuwanie elementów tablicy na procedurę. Najprostszy sposób postępowania polega na utworzeniu procedury bezparametrowej, zawierającej trzy linie programu, np.

```
procedure PrzesunLewo1;
begin
  ElementSkrajny := Porzadek[1];
  for j := 1 to 5 do Porzadek[j] := Porzadek[j + 1];
                                                {przesuwanie}
  Porzadek[6] := ElementSkrajny
end;
```

Procedura ta posłuży nam jako punkt wyjścia do utworzenia procedury uniwersalnej. Po pierwsze zmienne *j* oraz *ElementSkrajny* zadeklarujemy jako zmienne lokalne. Po drugie z tablicy uczynimy parametr formalny. Zauważ, że trzeba użyć słowa VAR, ponieważ będziemy zmieniać w procedurze wartości elementów tablicy będącej parametrem aktualnym (użytej w programie).

```
procedure PrzesunLewo2(var tablica: tab6intyp);
var j, ElementSkrajny: integer;
begin
  ElementSkrajny := tablica[1];
  for j := 1 to 5 do tablica[j] := tablica[j + 1];
                                                {przesuwanie}
  tablica[6] := ElementSkrajny
end;
```

Porządkowanie elementów tablicy wymaga przesuwania ich w różnym zakresie, a nie od początku do końca. Jeżeli po odczytaniu danych największy element znajduje się na miejscu 3, to pierwsze przesunięcie cykliczne w lewo powinno nastąpić pomiędzy wyrazem trzecim i szóstym. Wówczas największy wyraz (trzeci) trafi na pozycję szóstą, a czwarty, piąty i szósty zostaną przesunięte o jedno miejsce w lewo. W drugim kroku, wyszukuje się największy element wśród pierwszych pięciu i przesuwa go na miejsce piąte itd.

Zmiana treści procedury, żeby mogła ona przesuwać elementy w danym zakresie jest dość prosta. Zamiast stałych określających zakres przesuwania elementów należy wprowadzić parametry. Wymaga to jednak uwzględnienia w treści procedury różnych możliwych sytuacji spowodowanych nieodpowiednią wartością parametrów.

Zauważ, że punktem wyjścia jest procedura przesuwająca pięć elementów w lewo o jedną pozycję, a jeden w prawo o pięć pozycji. Można ją z łatwością przerobić na procedurę przesuwającą w lewo mniejszą liczbę elementów o jedną pozycję (np. tylko elementy 4 i 5) i jeden element w prawo o kilka pozycji (element 3 na miejsce 5), natomiast przesuwanie elementów w drugą stronę wymagałoby większych zmian. Jeżeli nie uczynimy tych większych zmian, to trzeba zabezpieczyć się przed taką wartością parametrów, przy której elementy tablicy miałyby być przesuwane o jedno miejsce w prawo, a pojedynczy element (o kilka pozycji) w lewo. Można to uczynić tak jak w pokazanej procedurze PRZESUN:

```
procedure Przesun(Skad, Dokad : integer; var tablica: tab6intyp);
var j, ElementSkrajny: integer;
begin
  if Skad < Dokad then
    begin
      ElementSkrajny := tablica[Skad];
      for j := Skad to Dokad - 1 do tablica[j] := tablica[j + 1];
      tablica[Dokad] := ElementSkrajny
    end
end;
```

Można też wzbogacić procedurę w drukowanie odpowiednich komunikatów (co możesz dodać sam).

Dysponując taką procedurą mógłbyś z łatwością powtarzać w programie głównym przesuwanie wszystkich elementów tablicy, między miejscami wskazanymi przez użytkownika, np.

## Program TP-30

```
program TP30;  {fragment programu glownego}
begin
                              {Czytanie danych}
  DrukTabl(Porzadek);
  repeat
    Write ('Podaj numer elementu, ktory chcesz przesunac ');
    Write ('( < 1   -  koniec) ');
    ReadLn (NumerSkad);
    if NumerSkad > 0 then
      begin
```

```
        Write ('Podaj numer pozycji, dokad chcesz go
                                              przesunac ');
        ReadLn (NumerDokad);
        Przesun(NumerSkad,NumerDokad,Porzadek);
        DrukTabl(Porzadek)
      end
  until NumerSkad < 1;
  Readln
end.
```

## Ćwiczenie 10-31

A. Napisz i uruchom program przesuwania elementów macierzy *Porzadek* oparty na podanych wskazówkach. Porównaj go z programem zawartym w pliku TP30.PAS

B. Uzupełnij swój program o badanie wprowadzanych danych, czy mieszczą się w rozsądnym zakresie, oraz ewentualnie o dokładniejsze komunikaty. Porównaj go z programem zawartym w pliku TP30P.PAS.

C. Rozbuduj swój program tak, aby był bardziej przejrzysty, czyniąc procedurami różne jego fragmenty funkcjonalne, w szczególności odczytywanie danych i towarzyszące tej czynności wypisywanie komunikatów i badanie warunków.

## Funkcja wyznaczająca pozycję największego elementu tablicy

Pozycję największego elementu w danej części tablicy, wskazywaną przez użytkownika w programie TP-30, można wyznaczać w programie. Uczyńmy to za pomocą funkcji, wykorzystując do jej utworzenia doświadczenia uzyskane podczas tworzenia programu TP-13, w którym była wyznaczana największa z wprowadzanych liczb.

```
function PozMax(nmax:integer; dane:tab6intyp):integer;
var
  licznik, ElementMax, PozycjaMax: integer;
begin
  ElementMax := dane[1];
  PozycjaMax := 1;
  for licznik := 2 to nmax do
   if dane[licznik] > ElementMax then
     begin
       ElementMax := dane[licznik];
       PozycjaMax := licznik
     end;
```

```
  PozMax := PozycjaMax;
end;
```

## Ćwiczenie 10-32

Zmodyfikuj swój program w taki sposób, żeby przesuwany element był wskazywany przez przedstawioną funkcję, a nie przez użytkownika. Uruchom go. Porównaj swój program z programem zawartym w pliku TP-PRZES.PAS

## Największy wspólny dzielnik

Program TP-18, w którym wyznaczałeś największy wspólny dzielnik i najmniejszą wspólną wielokrotność, możesz przekształcić na procedurę. Jeżeli ograniczysz się do wyznaczania jednej tylko wielkości, to możesz utworzyć funkcję. Zasadnicza część obliczeń dotyczyła wyznaczania dzielnika; wielokrotność była obliczana dopiero na jego podstawie za pomocą prostego wzoru. Utwórz więc funkcję NWDzielnik. Czy Twoja funkcja wygląda podobnie?

```
function NWDzielnik (m, n: integer):longint;
var
  ilorazm,ilorazn,czynnik,nwd: longint;
begin
  if (m < 1) or (n < 1) then Exit;
  czynnik := 2;
  ilorazm := m;
  ilorazn := n;
  nwd := 1;
  While (ilorazm > 1) and (ilorazn > 1) and (czynnik <= m) and
                                          (czynnik <= n) do
    if (ilorazm MOD czynnik = 0) and (ilorazn MOD
                                        czynnik = 0) then
      begin
        ilorazm := ilorazm DIV czynnik;
        ilorazn := ilorazn DIV czynnik;
        nwd := nwd * czynnik;
      end
    else
      czynnik := czynnik + 1;
  NWDzielnik := nwd;
end;
```

## Ćwiczenie 10-33

Napisz program służący do dodawania ułamków — realizujący zadanie 8-7 według algorytmu 10.

Wskazówka. Użyj w programie funkcji NWDZIELNIK. Możesz porównać swój program z programem zapisanym w pliku TPULAMKI.PAS.

## 10.11. Procedury graficzne

### 10.11.1. Wprowadzenie

Jeżeli w Twojej szkolnej pracowni komputerowej jest używany arkusz kalkulacyjny, to zapewne wygodniej jest korzystać z możliwości, które on oferuje. Samodzielne pisanie programów służących do tworzenia wykresów nie jest trudne, ale jeżeli wykres taki miałby być opisany i wyskalowany, to wszystko razem mogłoby okazać się dość pracochłonne. Niekiedy jednak wygodniej jest skorzystać z możliwości graficznych oferowanych przez język programowania, w szczególności wówczas, gdy na ekranie miałby być odtworzony ruch. Pełny opis możliwości graficznych systemu Turbo Pascal jest dość obszerny. Tu przedstawimy jedynie możliwości przykładowe.

Na ekran pracujący w trybie graficznym trzeba spojrzeć jako na swojego rodzaju prostokątną tablicę, której elementami są punkty (piksele). Liczba elementów zależy od użytej karty graficznej. Zazwyczaj w poziomie jest nie mniej niż 640 punktów, a w pionie nie mniej niż 348, z wyjątkiem komputerów wyposażonych w kartę CGA, która daje możliwość użycia tylko 200 punktów w pionie. Poszczególne punkty są numerowane od zera, a więc największy numer jest o jeden mniejszy od liczby punktów w danym kierunku.

Jeżeli Twój komputer zawiera kolorowy monitor, to każdemu punktowi możesz nadać kolor; jeżeli monochromatyczny, to możesz nadać stopień szarości. Nie zawsze masz jednak tu taką dowolność jak w przypadku tekstów, możesz bowiem dysponować np. tylko dwoma lub tylko czterema kolorami jednocześnie. Co więcej, przy większej liczbie kolorów możesz dysponować mniejszą liczbą punktów.

Funkcje graficzne są zawarte w module GRAPH. Procedura DETECTGRAPH rozpoznaje rodzaj zainstalowanej karty graficznej (sterownika) i tryb pracy karty.

```
detectgraph(sterownik, tryb);
```

Karcie VGA odpowiada sterownik nr 9, EGA — 3 (EGA64 — 4), CGA — 1, a karcie Hercules — 7. Interpretacja trybu zależy od karty, np. tryb nr 2 karty VGA oznacza użycie 16 kolorów przy rozdzielczości 640 na 480 punktów.

Tryb graficzny jest inicjowany wywołaniem procedury INITGRAPH. Ma ona trzy parametry: sterownik i tryb (oba typu *integer*) oraz parametr będący łańcuchem, wskazujący katalog dyskowy ze sterownikami ekranu