INFORMATYKA
KOMPUTERY
SYSTEMY

Dodatek „Żołnierza Wolności" Cena 200 zł

zeszyty programów komputerowych

nr 1 /88

NA KAŻDY KOMPUTER PASCAL

Gry

```
BEGIN
WSP: = pi/180
FOR i = ØTO 359 DO
BEGIN
END
```

SPIS TREŚCI

Wprowadzenie — str. 2

Podstawowe instrukcje i typy danych — str. 2

Złożone typy danych — str. 10

Procedury i funkcje — str. 15

Elementy grafiki komputerowej — str. 19

Dodatek (Jak uruchomić program?) — str. 23

Programy: I „Gra Hamiltona" — str. 25

„Wieża Hanoi" — str. 29

Opisy gier komputerowych:
Vegas Jackpot — str. 30
Alley Cat — str. 31

UWAGA! Następny zeszyt zawiera opis języka C — rewelacyjne narzędzie w rękach użytkowników mikrokomputerów.

„IKS" — dodatek „Żołnierza Wolności". Redaguje Wiesław Cetera (kierownik zespołu); Rada programowa: Krzysztof Chmarra, Romuald Głąb, Włodzimierz Gogołek, Janusz Janiec, Henryk Krasuski, Ireneusz Miernik, Ludwik Piela, Jacek Szaniawski. Adres redakcji: 00-950 Warszawa ul. Grzybowska 77, telefon centrali 20-12-61 w. 486. Telex 313664. Rękopisów nie zamówionych redakcja nie zwraca i zastrzega sobie prawo do skrótów. Nakładem: Wydawnictwa „Czasopisma Wojskowe", Warszawa ul. Grzybowska 77. Fotoskład i druk offsetowy — Wojskowe Zakłady Graficzne im. gen. dyw. A. Zawadzkiego.

U-40

Nr zam. 1117. Nr ind. 361682.

Wprowadzenie

Niniejsze opracowanie przeznaczone jest dla użytkowników mikrokomputerów, którzy pragną poszerzyć swoją wiedzę w zakresie samodzielnego programowania w języku PASCAL.

Wprowadzane wiadomości teoretyczne autorzy ilustrują prostymi przykładami zadań, które zawierają rozwiązania praktyczne. Przykłady te decydują o właściwym opanowaniu przedstawionych zagadnień i dostosowane są do możliwości percepcyjnych uczniów szkół średnich oraz wyższych klas szkół podstawowych.

Opracowanie składa się z 4 rozdziałów:

1. Podstawowe instrukcje i typy danych
Rozdział obejmuje przykłady rozwiązań prostych zadań obliczeniowych, uwzględniających przede wszystkim materiał możliwy do opanowania przez uczniów szkół podstawowych.

2. Złożone typy danych
Zawiera przykłady zapisu zbiorów danych z uwzględnieniem możliwości ich przychowywania w pamięci zewnętrznej.

3. Procedury i funkcje
Jest to wstęp do programowania zaawansowanego, w którym wykorzystuje się logiczne wyodrębnione fragmenty programów.

Elementy grafiki komputerowej
Obejmuje przykłady wyświetlania tekstów i obrazów.

Wszystkie zaprezentowane programy zostały uruchomione w systemie TURBO PASCAL na mikrokomputerze AMSTRAD CPC 61128, pracującym pod nadzorem CP/M PLUS. Podstawowe zasady pracy w systemie TURBO PASCAL zawiera dodatek. Przedstawione programy, poza tymi, które obejmują elementy grafiki, mogą być uruchamiane także na mikrokomputerach serii IBM PC. Systematyczne ujęcie materiału powinno sprzyjać opanowaniu podstawowych właściwości języka PASCAL bez

PODSTAWOWE INSTRUKCJE I TYPY DANYCH

1



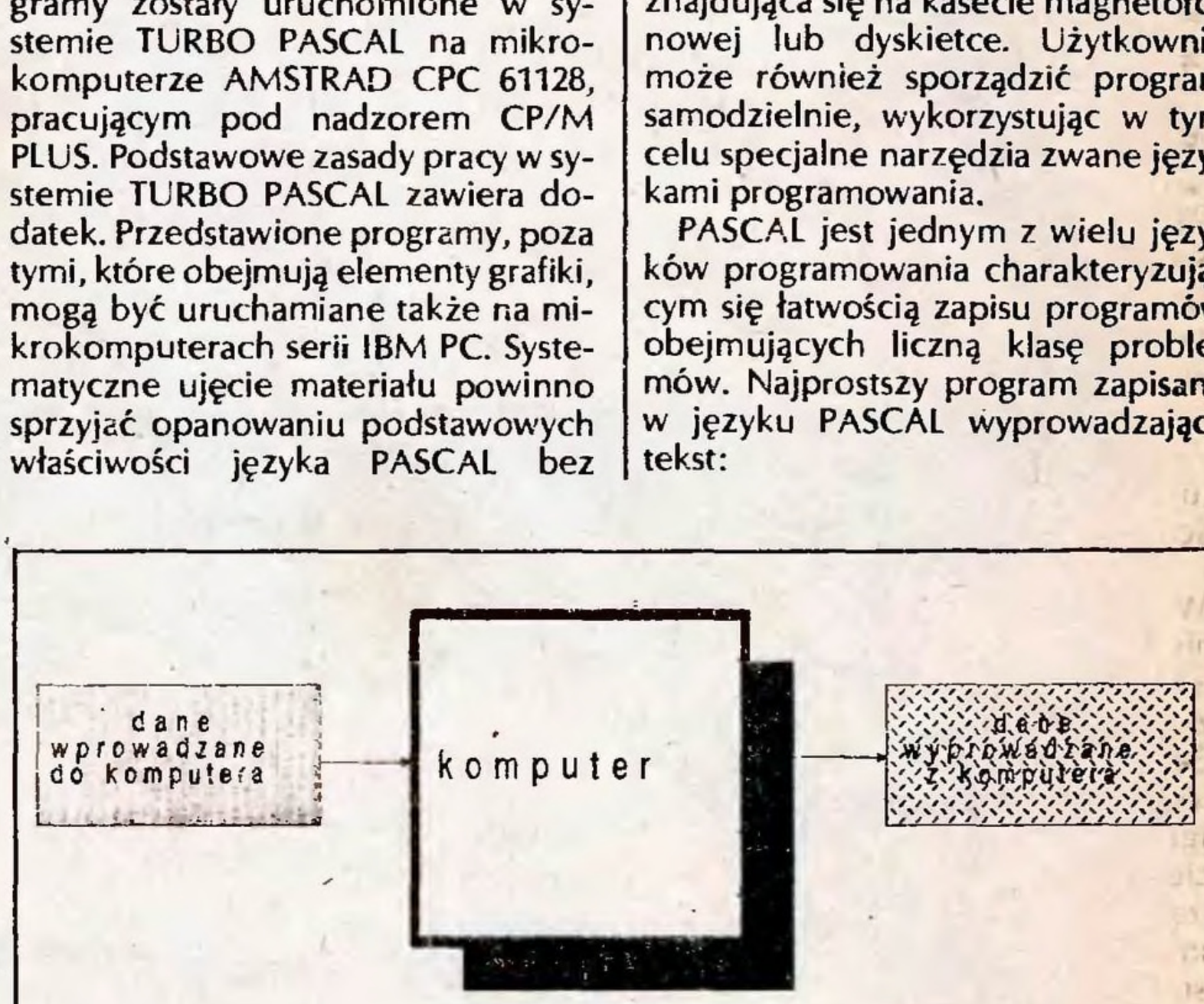
względu na komputer, z którym jest on związany.

Komputer jest narzędziem przystosowanym do przetwarzania informacji. Informacje umieszczone na odpowiednich nośnikach zwane są danymi. Stąd też często mówi się o tym, że komputer umożliwia przetwarzanie danych. Proces użytkowania komputera przedstawia rysunek 1.

Dane mogą być wprowadzane do komputera jako tekst w języku angielskim, natomiast wyprowadzane z komputera jako ten sam tekst przetłumaczony na język polski.

W tym przypadku przetwarzanie danych przez komputer polega na tłumaczeniu tekstu. Ten sam komputer może być wykorzystywany w innych celach np. do obliczania pola kwadratu na podstawie wyprowadzonej długości boku. W każdym z obydwu przypadków komputer realizuje inne czynności. Sposób postępowania komputera określa przepis zawarty w tzw. programie. Program ten musi być wprowadzony do komputera przed rozpoczęciem wymaganych czynności. Takim programem może być dowolna gra komputerowa znajdująca się na kasecie magnetofonowej lub dyskietce. Użytkownik może również sporządzić program samodzielnie, wykorzystując w tym celu specjalne narzędzia zwane językami programowania.

PASCAL jest jednym z wielu języków programowania charakteryzującym się łatwością zapisu programów obejmujących liczną klasę problemów. Najprostszy program zapisany w języku PASCAL wyprowadzający tekst:



RYS.1. Proces użytkowania komputera

rozpoczynamy naukę języka *PASCAL* na monitor zamieszczono w przykładzie 1.

Przykład 1

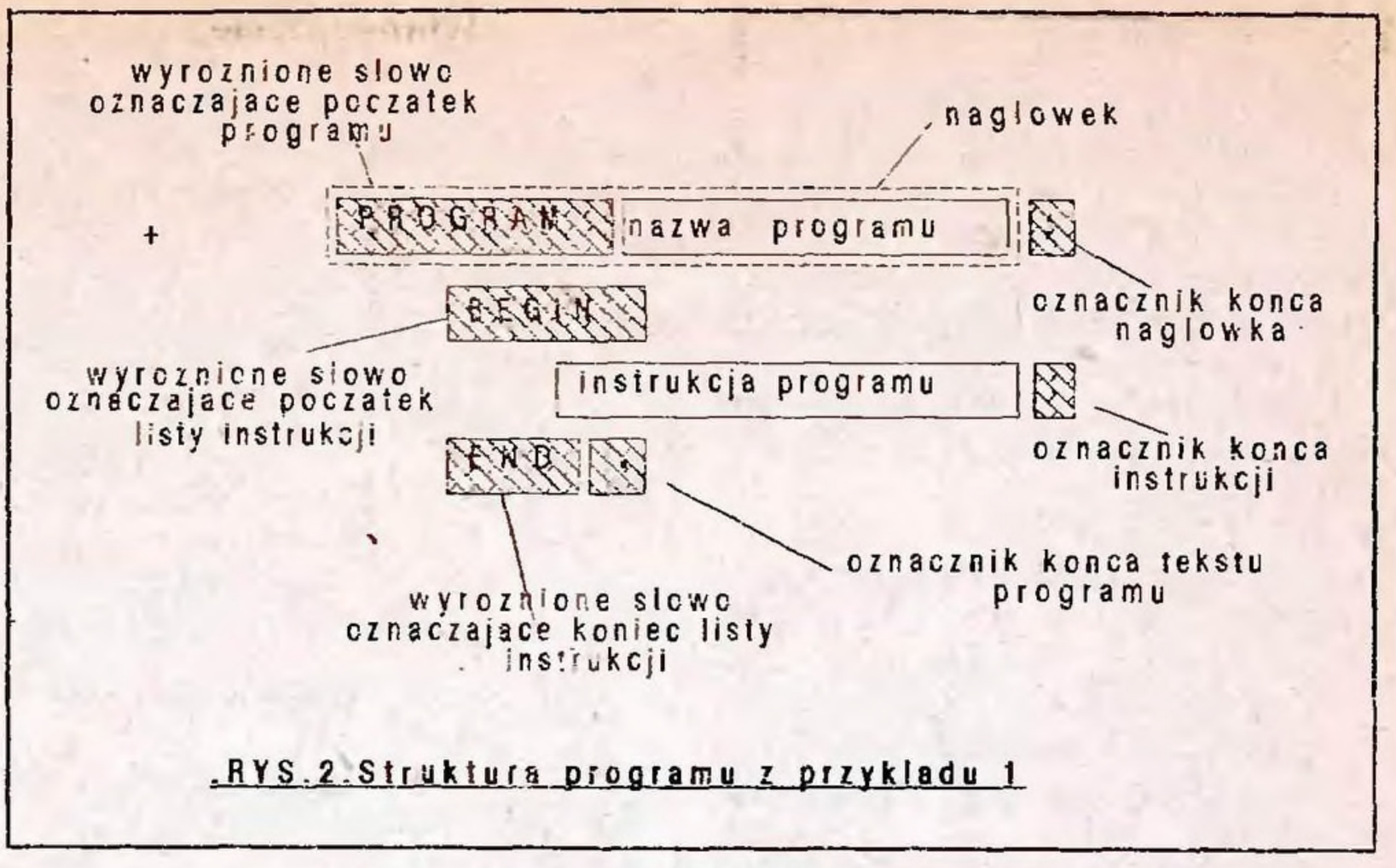
```
PROGRAM prog1;
 BEGIN
  WRITE('rozpoczynamy naukę języka PASCAL');
END.
```

Program posiada strukturę przedstawioną na rysunku 2.

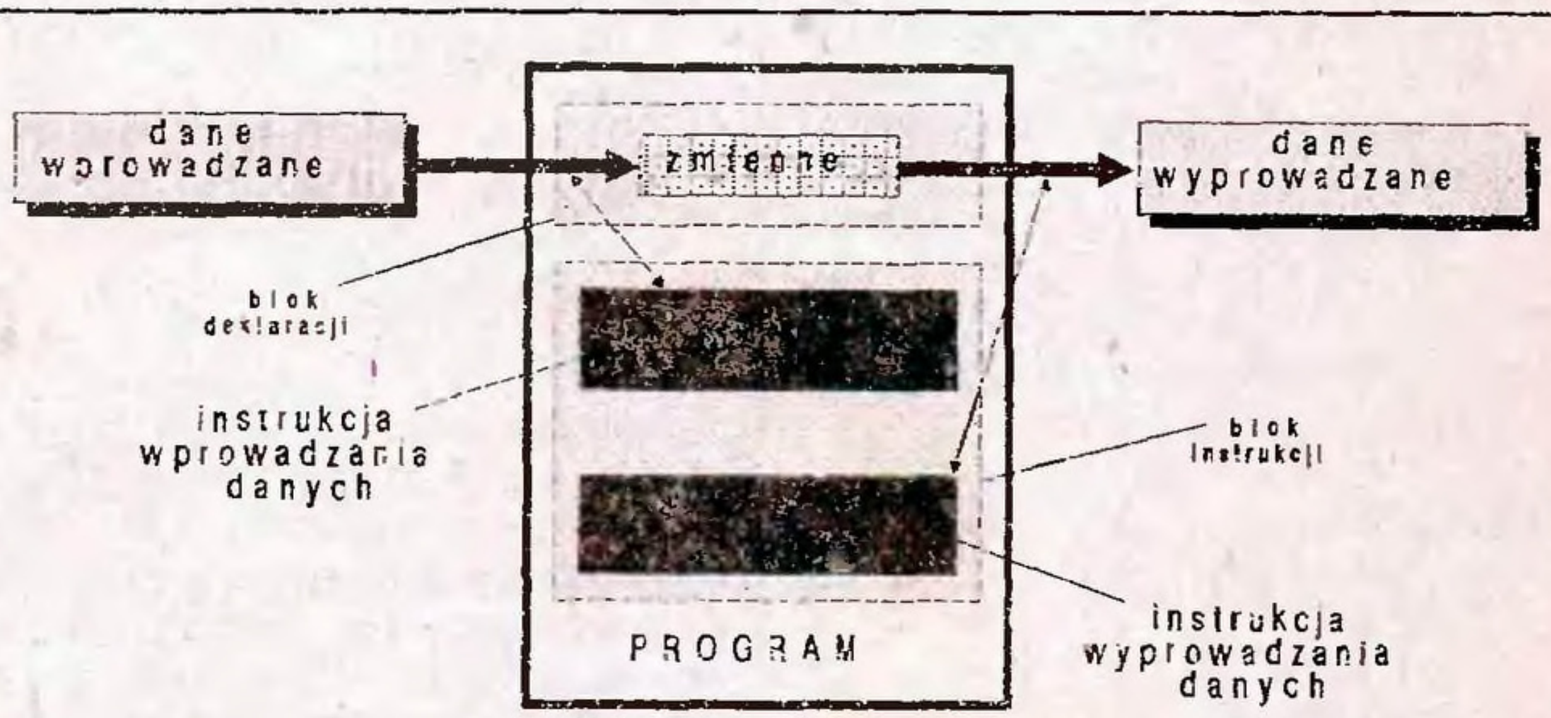
Wszystkie wyróżnione słowa (zapisane wielkimi literami) zwane są **słowami kluczowymi.** Każde z nich ma ściśle określone znaczenie.

Podstawowym elementem programu o nazwie **prog1** jest instrukcja **WRITE** powodująca wyprowadzenie na monitor tekstu ujętego w apostrofy. Program ten opisuje szczególny przypadek przetwarzania danych, nie wymaga bowiem wyprowadzania żadnych danych wejściowych.

Najprostszy program uwzględniający wprowadzanie i wyprowadzanie danych w niezmienionej postaci, musi zawierać co najmniej dwie instrukcje, z których jedna wprowadza dane, a druga powoduje ich wyprowadzenie. Pomiędzy wprowadzeniem a wyprowadzeniem dane muszą być gdzieś przechowywane. Struktura programu powinna zatem uwzględniać istnienie obiektów umożliwiających przechowanie danych. Obiekty te zwane są zmiennymi. Zasadę konstruowania takiego programu przedstawia rysunek 3, a sam program został zamieszczony w przykładzie 2.



RYS.2.Struktura programu z przykladu 1



RYS.3.Ilustracja zasady konstrukcji programu wprowadzania i wyprowadzania danych

Przykład 2

```
PROGRAM prog2;
 VAR
  tekst:string[5];
 BEGIN
  READ(tekst);
  WRITE(tekst);
 END.
```

Strukturę programu przedstawiono na rys. 4.

W programie występuje deklaracja zmiennej, tj. zapis określający:

— nazwę zmiennej (*tekst*),

— typ zmiennej (*string* [5]) identyfikujący zakres dopuszczalnych wartości, jakie może przyjmować zmienna.

W programie **prog2** wartością zmiennej *tekst* może być ciąg dowolnych znaków o długości nie większej niż 5. Przykładowo ciągi *abcde*, xy, 1234 są poprawnymi wartościami zmiennej *tekst* w przeciwieństwie do ciągu 123456, który jest zbyt długi. W liście instrukcji program prog. 2 pierwszą instrukcją jest **READ** z parametrem *tekst*. W wyniku jej wykonania następuje wczytanie ciągu znaków (o długości nie przekraczającej 5) do zmiennej *tekst*. Wyprowadzenie wczytanego tekstu realizuje znana już instrukcja WRITE, której parametrem jest w tym przypadku zmienna *tekst*.



Rys.4.Struktura programu z przykladu 2

Program **prog2** wykonany dla ciągu *abcd* (zakończenie wprowadzania ciągu następuje po wciśnięciu klawisza **ENTER**) powoduje wyświetlenie tekstu *abcd*. Jest on wyświetlany bezpośrednio po ciągu wejściowym. W efekcie na ekranie monitora pozostaje ciąg *abcdabcd*. Chcąc oddzielić te ciągi możemy wyświetlić ciąg wyprowadzany w nowej linii. Efekt ten uzyskamy wstawiając instrukcję **WRITELN** pomiędzy instrukcje READ I WRITE (przykład 3).

Przykład 3

```
PROGRAM prog3;
 VAR
  tekst:string[5];
 BEGIN
  READ(tekst);
  WRITELN;
  WRITE(tekst);
 END.
```

Podane przykłady dotyczyły wprowadzania i wyprowadzania łańcuchów znaków, PASCAL umożliwia również wprowadzanie danych traktowanych jako liczby. Program zamieszczony w przykładzie 4 pozwala na wprowadzenie dwóch liczb rzeczywistych (liczby dziesiętne, w których część ułamkową — jeśli występuje — oddziela się od części całkowitej kropką zamiast przecinka) i ich wyprowadzenie.

W programie wykorzystano instrukcje z przykładu 3. Programy różnią się:

— deklaracjami zmiennych — zmienne x i y są zadeklarowane jako liczby rzeczywiste **(real),**
— postacią instrukcji WRITE, w której oprócz nazwy wyprowadzanej zmiennej podano sumaryczną ilość znaków przeznaczonych na zapis liczby *(6)* oraz ilość znaków przeznaczonych na część ułamkową *(2).*

W uzupełnieniu należy dodać, że sumaryczna ilość znaków liczby obejmuje kropkę dziesiętną oraz znak liczby (wyprowadzany tylko w przypadku liczb ujemnych).

Przykład 4

```
PROGRAM prog4;
 VAR
  x:real;
  y:real;
BEGIN
 READ(x);
 READ(y);
 WRITELN;
 WRITE('x=');
 WRITE(x:6:2);
 WRITELN;
 WRITE('y=');
 WRITE(y:6:2);
END.
```

Program **prog4** wykonano dla liczb x = 210,85 i y = —20 uzyskując wyniki:

x = 210.85
y = 20.00

Program **prog4** pomimo iż realizuje bardzo proste funkcje, jest stosunkowo obszerny. W celu skrócenia zapisu można wykorzystać następujące właściwości języka:

— dla zmiennych jednakowego typu, nazwa typu może być podana po wyliczeniu nazw tych zmiennych,
— w ramach jednej instrukcji READ można wprowadzić więcej niż jedną daną, przy czym nie muszą być one tego samego typu,
— w ramach jednej instrukcji WRITE można wyprowadzić więcej niż jedną daną (bez względu na jej typ).

Właściwości te wykorzystano w programie **prog5** (przykład 5) realizującym te same funkcje co program **prog4.**



Rys.5.Struktura prostego programu

Przykład 5

```
PROGRAM prog5;
 VAR
  x,y:real;
BEGIN
 READ(x,y);
 WRITELN;
 WRITELN('x=',x:6:2);
 WRITE('y=',y:6:2);
END.
```

W dotychczas podanych przykładach działanie programów ograniczało się do wprowadzania i wyprowadzania danych. Podstawowym zadaniem komputera jest jednak przetwarzanie danych realizowanych za pomocą odpowiednich instrukcji języka PASCAL.

Typową strukturę programu uwzględniającą wprowadzenie, przetwarzanie i wyprowadzanie danych przedstawiono na rysunku 5.

Podstawową instrukcją przetwarzania danych jest instrukcja przypisania przyjmująca formę:

nazwa zmiennej := *wyrażenie*

identyfikator instrukcji

Instrukcja ta powoduje przypisanie zmiennej, występującej po lewej stronie identyfikatora instrukcji, nowej wartości, będącej wynikiem obliczenia wartości wyrażenia występującego po prawej stronie znaku identyfikatora instrukcji.

W odniesieniu do zmiennych rzeczywistych wyrażenia w języku PASCAL konstruowane są tak jak wyrażenia arytmetyczne.

Różnice dotyczą wyłącznie postaci niektórych operatorów oraz zapisu ułamków. Sposób wyliczania wartości wyrażenia określony jest wagą operatorów:

— wyrażenia w nawiasach obliczane są w pierwszej kolejności,
— mnożenie (* zamiast .) oraz dzielenie (/ zamiast :) wykonywane są w kolejności ich występowania przed wykonaniem dodawania i odejmowania,
— dodawanie (+) i odejmowanie (—) wykonywane są w kolejności ich występowania.

Ułamki traktowane są jako operacje dzielenia licznika przez mianownik.

Proste programy zamieszczone w przykładach 6—11 obejmują kolejno obliczenie:

— sumy, różnicy, iloczynu i ilorazu dwóch liczb rzeczywistych,
— wartości sinusa, cosinusa i tangensa jednego z kątów trójkąta prostokątnego o podanych bokach,
— pola prostokąta o podanych bokach,
— pola trapezu o podanych podstawach i wysokości,
— objętości kuli o podanym promieniu,
— objętości stożka o podanym promieniu podstawy i zadanej wysokości.

Przykład 6

```
PROGRAM prog6;
 VAR
  x,y,suma,roznica,
  iloczyn,iloraz:real;
BEGIN
 READ(x,y);
 WRITELN;
 WRITELN('pierwsza liczba =',x:8:4);
 WRITELN('druga liczba =',y:8:4);
 suma:=x+y;
 roznica:=x-y;
 iloczyn:=x*y;
 iloraz:=x/y;
 WRITELN('suma =',suma:8:4);
 WRITELN('roznica =',roznica:8:4);
 WRITELN('iloczyn =',iloczyn:8:4);
 WRITE('iloraz =',iloraz:8:4);
END.
```

Przykład 7

```
PROGRAM prog7;
 VAR
  a,b,c,sinus,cosinus,tangens:real;
BEGIN
 READ(a,b,c);
 WRITELN;
 WRITELN('wczytane boki trojkata',
         ' prostokatnego:');
 WRITELN('a =',a:3:0,', b =',b:3:0,
         ', c =',c:3:0);
```

```
  sinus:=b/c;
  cosinus:=a/c;
  tangens:=b/a;
  WRITELN;
  WRITELN('sinus kata alfa = ',sinus:6:4);
  WRITELN('cosinus kata alfa = ',cosinus:6:4);
  WRITE('tangens kata alfa = ',tangens:6:4);
END.
```

Przykład 8

```
PROGRAM prog8;
 VAR
  a,b,pole:real;
 BEGIN
  WRITELN;
  WRITELN('OBLICZENIE POLA PROSTOKATA:');
  WRITELN('podaj dlugosci bokow');
  READ(a,b);
  pole:=a*b;
  WRITELN;
  WRITELN('pole prostokata o bokach');
  WRITELN(' a =',a:6:3,', b =',b:6:3);
  WRITE('wynosi ',pole:8:3);
 END.
```

Przykład 9

```
PROGRAM prog9;
 VAR
  a,b,h,pole:real;
 BEGIN
  WRITELN;
  WRITELN('OBLICZENIE POLA TRAPEZU:');
  WRITELN('podaj dlugosci:');
  WRITELN('   -dolnej podstawy,');
  WRITELN('   -gornej podstawy,');
  WRITELN('   -wysokosc.');
  READ(a,b,h);
  pole:=0.5*(a+b)*h;
  WRITELN;
  WRITELN('pole trapezu o podstawach');
  WRITELN(' a =',a:6:3,', b =',b:6:3);
  WRITELN('i wysokosci h =',h:6:3);
  WRITE('wynosi ',pole:8:3);
 END.
```

Przykład 10

```
PROGRAM prog10;
 VAR
  r,objetosc:real;
 BEGIN
  WRITELN;
  WRITELN('OBLICZENIE OBJETOSCI KULI:');
  WRITELN('podaj promien kuli');
  READ(r);
  objetosc:=4/3*pi*r*r*r;
       {pi - stala dostepna w PASCAL-u}
  WRITELN;
  WRITELN('objetosc kuli o promieniu');
  WRITELN(' r =',r:6:3);
  WRITE('wynosi ',objetosc:10:3);
 END.
```

Przykład 11

```
PROGRAM prog11;
 VAR
  r,h,objetosc:real;
 BEGIN
  WRITELN;
  WRITELN('OBLICZENIE OBJETOSCI STOZKA:');
  WRITELN('podaj promien podstawy');
  WRITELN('i wysokosc stozka');
  READ(r,h);
  objetosc:=pi*r*r*r/3;
  WRITELN;
  WRITELN('objetosc stozka o promieniu');
  WRITELN('podstawy r =',r:6:3);
  WRITELN('i wysokosci h =',h:6:3);
  WRITE('wynosi ',objetosc:8:3);
 END.
```

Dotychczas poznane instrukcje umożliwiają już wykonanie prostych obliczeń na wprowadzonych danych oraz prezentację wyników. Na ogół jednak to nie wystarcza, ponieważ wykonanie takich działań często zależy do spełnienia określonych warunków. Na przykład wyznaczenie wartości bezwzględnej liczby zgodnie z definicją:

$$|x| = \begin{cases} x \text{ dla } x \geq 0 \\ -x \text{ dla } x < 0 \end{cases}$$

wymaga porównania wartości liczby z zerem.

Podstawową instrukcją umożliwiającą wykonanie działań uzależnionych od spełnienia określonych warunków jest instrukcja alternatywy przyjmująca postać:

IF *warunek* **THEN**

instrukcja 1

ELSE

instrukcja 2

Warunek jest wyrażeniem logicznym tzn. takim, które w danej sytuacji (dla określonych wartości zmiennych w nim występujących) może być spełnione lub nie. Jeżeli warunek jest spełniony to następuje wykonanie *instrukcji 1* (a *instrukcja 2* nie jest w ogóle wykonywana). W przypadku przeciwnym wykonywana jest *instrukcja 2* (*instrukcja 1* nie jest wtedy wykonywana).

Program obliczania wartości bezwzględnej zadanej liczby zamieszczono w przykładzie 12.

Przykład 12

```
PROGRAM prog12;
 VAR
  x:real;
 BEGIN
  READ(x);
  WRITELN;
  WRITELN('wartosc bezwzgledna liczby');
  WRITELN('      ',x:8:4);
  IF  x>=0 THEN
    x:=x
  ELSE
    x:=-x;
  WRITE('wynosi',x:8:4);
 END.
```

Występujący w programie warunek

$$x >= 0$$

jest spełniony tylko wtedy, gdy x ma wartość nieujemną. Taką interpretację warunku wskazuje operator relacyjny >= (większy lub równy). W warunkach mogą występować ponadto następujące operatory relacyjne: < (mniejszy), <= (mniejszy lub równy), = (równy), > (większy), <> (różny).

Wyrażenia występujące w przykładzie 12 zawierają nazwy tych samych zmiennych po obu stronach operatora np:

$$x := -x$$

W takim przypadku przy obliczaniu wartości wyrażenia brana jest pod uwagę wartość zmiennej x przed wykonaniem instrukcji przypisania. Wyznaczona wartość wyrażenia jest następnie przypisywana zmiennej x jako nowa wartość (poprzednia wartość zostaje zniszczona).

Ze składni instrukcji alternatywy wynika, że zarówno w przypadku spełnienia, jak i niespełnienia warunku wykonywana jest pojedyncza instrukcja. W języku PASCAL za pojedynczą instrukcję przyjmuje się także instrukcję złożoną — sekwencję instrukcji ograniczoną słowami kluczowymi **BEGIN—END**:

BEGIN
instrukcja 1;
instrukcja 2;
instrukcja 3;
END

W przykładzie 13 zamieszczono zmodyfikowaną wersję programu z przykładu 7, w którym wyznaczanie sinusa, cosinusa i tangensa (zapisane w postaci instrukcji złożonej) jest wykonywane tylko wtedy, gdy trójkąt o podanych bokach jest trójkątem prostokątnym.

Przykład 13

```
PROGRAM prog13;
 VAR
  a,b,c,sinus,cosinus,tangens:real;
 BEGIN
  READ(a,b,c);
  WRITELN;
  WRITELN('wczytane boki trojkata');
  WRITELN('a =',a:3:0,', b =',b:3:0,
          ', c =',c:3:0);
  IF (a*a+b*b)=(c*c) THEN
   BEGIN
    sinus:=b/c;
    cosinus:=a/c;
    tangens:=b/a;
    WRITELN;
    WRITELN('sinus kata alfa = ',
            sinus:6:4);
    WRITELN('cosinus kata alfa = ',
            cosinus:6:4);
    WRITE('tangens kata alfa = ',
            tangens:6:4);
   END
  ELSE
   WRITE('nie tworza trojkata ',
         'prostokatnego');
END.
```

Bardziej złożony program, ilustrujący wykorzystanie instrukcji alternatywy, zamieszczono w przykładzie 14. Program ten umożliwia obliczanie pierwiastków rzeczywistych trójmianu kwadratowego. W poprzednich przykładach programista określał postać wyprowadzanych liczb. W przypadku, gdy programiście nie zależy na narzuceniu formatu wyprowadzanych liczb może wykorzystać postać standardową, która dla liczb rzeczywistych jest następująca:

uu#.##########E☆##
dla liczb większych lub równych zero,

u — #.##########E☆##
dla liczb mniejszych od zera.

gdzie:

u — *reprezentuje spację,*
— *reprezentuje cyfrę dziesiętną,*
☆ — *reprezentuje znak* + lub —

Wartość tak wprowadzonej liczby należy odczytać jako c. 10^m

gdzie:

c — *liczba znajdująca się przed literą E* (cecha),
m — *liczba znajdująca się za literą E* (mantysa).

Wartość zmiennej wyprowadzona jest w formacie standardowym, gdy w instrukcji WRITE występuje wyłącznie nazwa zmiennej.

Przykład 14

```
PROGRAM prog14;
 VAR
  a,b,c,
  x1,x2,
  delta:real;
 BEGIN
  WRITELN;
  WRITELN('podaj wspolczynniki trojmianu');
  READ(a,b,c);
  IF a=0 THEN
   WRITELN(' niedopuszczalna wartosc ',
           'wspolczynnika a')
ELSE
 BEGIN
  WRITELN;
  WRITELN('dla wspolczynnikow: a =',a);
  WRITELN('                    b =',b);
  WRITELN('                    c =',c);
  delta:=b*b-4*a*c;
  IF delta<0 THEN
   WRITELN('brak pierwiastkow rzeczywi',
           'stych')
  ELSE
   IF delta=0 THEN
    BEGIN
     x1:=-b/(2*a);
     WRITELN('trojmian ma jeden pierwi',
             'astek');
     WRITE('podwojny x =',x1);
    END
   ELSE
    BEGIN
     x1:=(-b-SQRT(delta))/(2*a);
     x2:=(-b+SQRT(delta))/(2*a);
     WRITELN('trojmian ma dwa pierwia',
             'stki: x1 =',x1);
     WRITE(' ':29,'x2 =',x2);
    END;
  END;
END.
```

Należy dodać, że wszystkie wprowadzane liczby rzeczywiste mogą być podane w postaci liczb dziesiętnych z kropką (np. 10.35) lub w postaci standardowej (np. 10.35E+0, 1.035E+1, 0.11035E+2, 1035E—2).

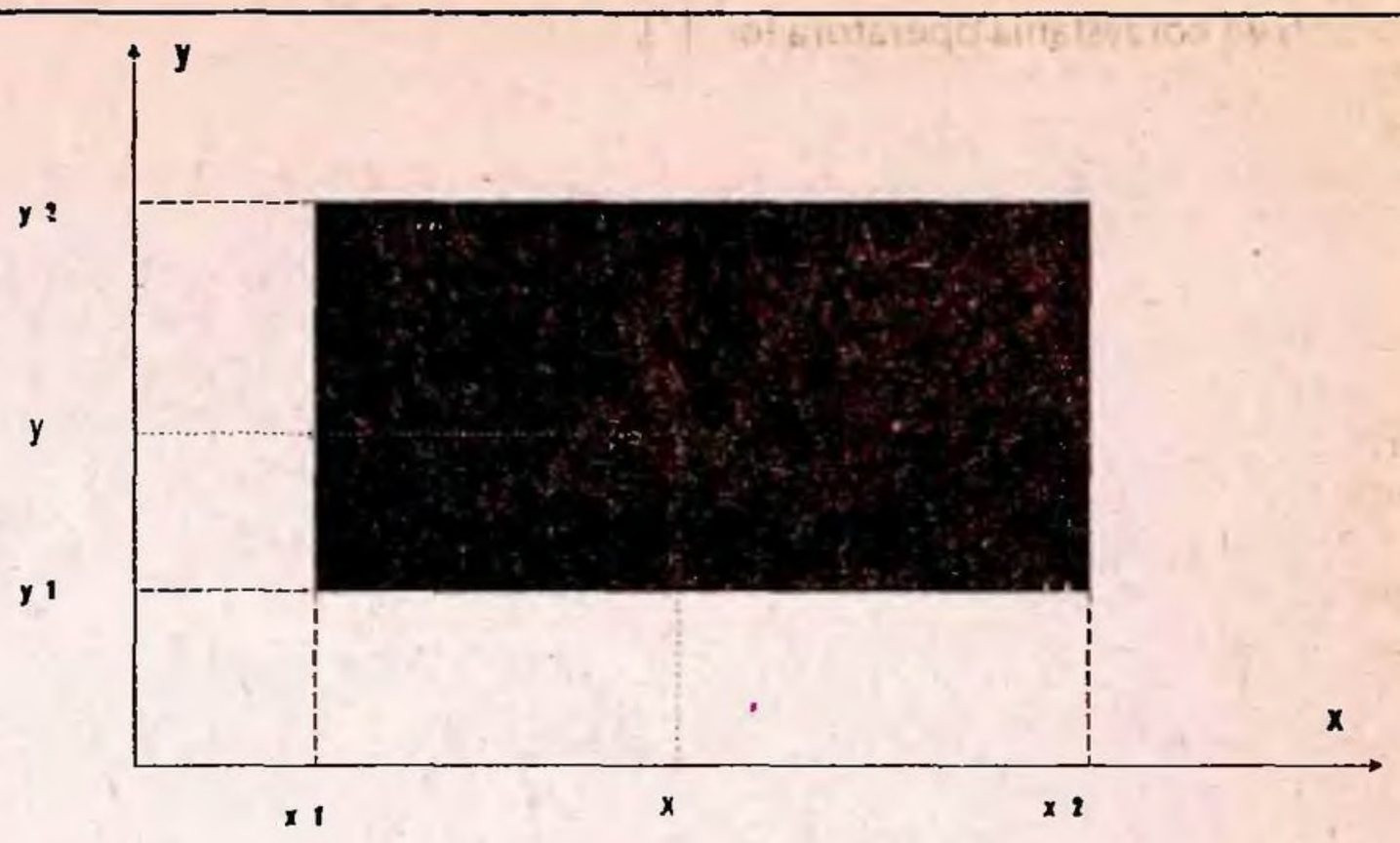
Instrukcja alternatywy zakłada wykonanie różnych akcji w przypadku spełnienia i niespełnienia warunku. Niejednokrotnie jednak konstrukcja programu wymaga wykonania jakiejś akcji tylko w przypadku spełnienia warunku. Wygodniej jest wtedy wykorzystać uproszczoną postać instrukcji alternatywy, zwaną instrukcją warunkową:

IF *warunek* **THEN** *instrukcja 1*

Instrukcja 1 wykonywana jest tylko w przypadku spełnienia warunku. Przykłady wykorzystania instrukcji warunkowej zamieszczono w programach **prog15** i **prog16** (przykłady 15 i 16).

Przykład 15

```
PROGRAM prog15;
 VAR
  x:real;
 BEGIN
  READ(x);
  WRITELN;
  WRITELN('wartosc bezwzgledna liczby');
  WRITELN('      ',x:8:4);
  IF  x<0 THEN
    x:=-x;
  WRITE('wynosi',x:8:4);
 END.
```



Rys.6.Ilustracja do przykładu 17

Przykład 16

```
PROGRAM prog16;
 VAR
  a,b,h,r,pole,objetosc:real;
  nrw:integer;
 BEGIN
  WRITELN;
  WRITELN('PROGRAM UMOZLIWIA OBLICZENIE:');
  WRITELN(' 1 - POLA PROSTOKATA,');
  WRITELN(' 2 - POLA TRAPEZU,');
  WRITELN(' 3 - OBJETOSCI KULI,');
  WRITELN(' 4 - OBJETOSCI STOZKA.');
  WRITELN('podaj numer wariantu');
  WRITELN;
  READLN(nrw);
  IF nrw=1 THEN
   BEGIN
    WRITELN;
    WRITELN('OBLICZENIE POLA PROSTOKATA:');
    WRITELN('podaj dlugosci bokow');
    READ(a,b);
    pole:=a*b;
    WRITELN;
    WRITELN('pole prostokata o bokach');
    WRITELN(' a =',a:6:3,', b =',b:6:3);
    WRITE('wynosi ',pole:8:3);
   END;
  IF nrw=2 THEN
   BEGIN
    WRITELN;
    WRITELN('OBLICZENIE POLA TRAPEZU:');
    WRITELN('podaj dlugosci:');
    WRITELN('   -dolnej podstawy,');
    WRITELN('   -gornej podstawy,');
    WRITELN('   -wysokosc.');
    READ(a,b,h);
    pole:=0.5*(a+b)*h;
    WRITELN;
    WRITELN('pole trapezu o podstawach');
    WRITELN(' a =',a:6:3,', b =',b:6:3);
    WRITELN('i wysokosci h =',h:6:3);
    WRITE('wynosi ',pole:8:3);
   END;
  IF nrw=3 THEN
   BEGIN
    WRITELN;
    WRITELN('OBLICZENIE OBJETOSCI KULI:');
    WRITELN('podaj promien kuli');
    READ(r);
    objetosc:=4/3*pi*r*r*r;
    WRITELN;
    WRITELN('objetosc kuli o promieniu');
    WRITELN(' r =',r:6:3);
    WRITE('wynosi ',objetosc:10:3);
   END;
  IF nrw=4 THEN
   BEGIN
    WRITELN;
    WRITELN('OBLICZENIE OBJETOSCI STOZKA:');
    WRITELN('podaj promien podstawy');
    WRITELN('i wysokosc stozka');
    READ(r,h);
    objetosc:=pi*r*r/3;
    WRITELN;
    WRITELN('objetosc stozka o promieniu');
    WRITELN('podstawy r =',r:6:3);
    WRITELN('i wysokosci h =',h:6:3);
    WRITE('wynosi ',objetosc:8:3);
   END;
 END.
```

W przykładzie 15 wystąpiła potrzeba zmiany operatora relacyjnego, zastosowanego w programie z przykładu 12. Operator $\geq$ zastąpiono operatorem $<$ w celu umieszczenia instrukcji

$$x := -x$$

w odpowiednim miejscu. Ten sam efekt można uzyskać stosując jeden z operatorów logicznych tj. operator negacji **NOT.** Instrukcja warunkowa w programie z przykładu 15 przyjęłaby wówczas postać:

IF NOT $x = 0$ **THEN** $x := -x$

Oprócz operatora NOT w PASCAL-u dostępne są ponadto operatory logiczne **OR** (logiczne *lub*) oraz **AND** (logiczne *i*) umożliwiające składanie warunków złożonych z warunków prostszych. Sposób wykorzystania operatora logicznego AND przedstawiono w przykładzie 17 zawierającym program sprawdzania czy punkt o zadanych współrzędnych leży w polu prostokąta określonego jak na rysunku 6.

Przykład 17

```
PROGRAM prog17;
 VAR
  x1,y1,x2,y2,
  x,y:real;
 BEGIN
  WRITELN;
  WRITELN('PODAJ WSPOLRZEDNE PUNKTOW:');
  WRITELN(' (x1,y1), (x2,y2), (x,y)');
  READ(x1,y1,x2,y2,x,y);
  WRITELN;
  WRITELN('PUNKT O WSPOLRZEDNYCH');
  WRITELN('x=',x,' y=',y);
  IF (x>=x1) AND (x<=x2) AND (y>=y1)
     AND (y<=y2) THEN
   WRITE('LEZY W POLU PROSTAKATA')
  ELSE
   WRITE('NIE LEZY W POLU PROSTOKATA');
 END.
```

Sposób wykorzystania operatora logicznego OR zamieszczono w przykładzie 18, zawierającym program obliczania wartości funkcji

$$f(x) = \begin{cases} 2 \text{ dla } x<-2 \text{ lub } x>2 \\ 2x \text{ dla } -2<x<2 \end{cases}$$

której przebieg zobrazowano na rysunku 7.

Przykład 18

```
PROGRAM prog18;
 VAR
  x,w:real;
 BEGIN
  WRITELN;
  WRITELN('PODAJ WARTOSC x');
  READ(x);
  WRITELN;
  WRITELN('WARTOSC FUNKCJI DLA x=',x);
  IF (x<-2) OR (x>2) THEN
   w:=5
  ELSE
   w:=2*x;
  WRITE('WYNOSI ',w);
END.
```

Podobnie jak w przypadku wyrażeń arytmetycznych przy obliczaniu warunków (stwierdzaniu, czy warunki są spełnione) obowiązują reguły pierwszeństwa realizacji. Są one następujące:

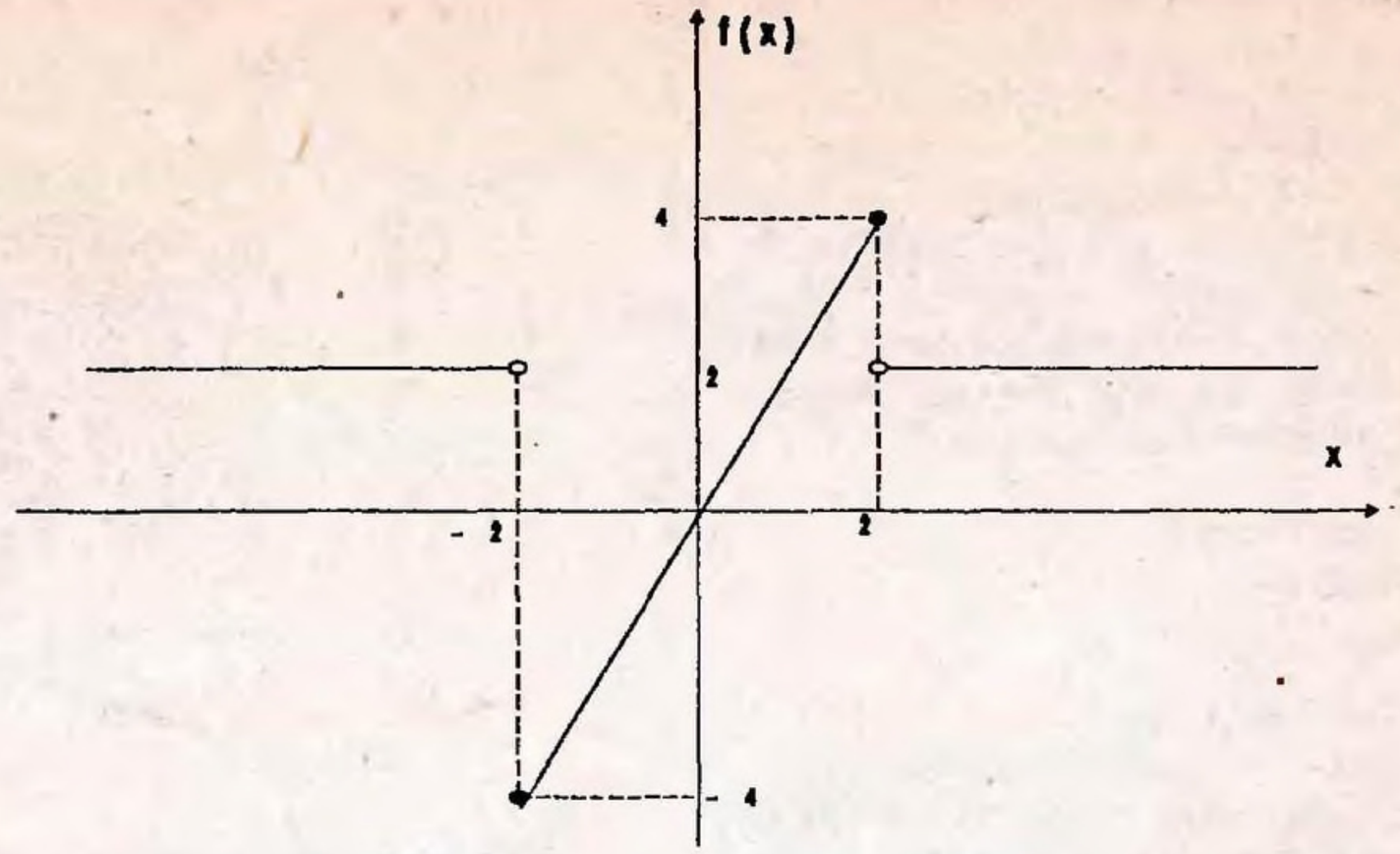
— *warunki w nawiasach obliczane są w pierwszej kolejności,*
— *najwyższy priorytet spośród operatorów logicznych ma operator NOT,*
— *operator AND ma pierwszeństwo przed operatorem OR.*

Wszystkie przytoczone dotychczas programy charakteryzują się możliwością jednorazowego wykonania opisanych w nich akcji. Powtarzanie tych akcji jest możliwe jedynie poprzez kolejne wykonania programów. Powtórzenie tych samych akcji przy jednokrotnym uruchomieniu programu byłoby możliwe poprzez powielenie wszystkich instrukcji opisujących daną akcję. Jest to działanie nieefektywne, ponieważ liczba żądanych powtórzeń często nie jest znana.

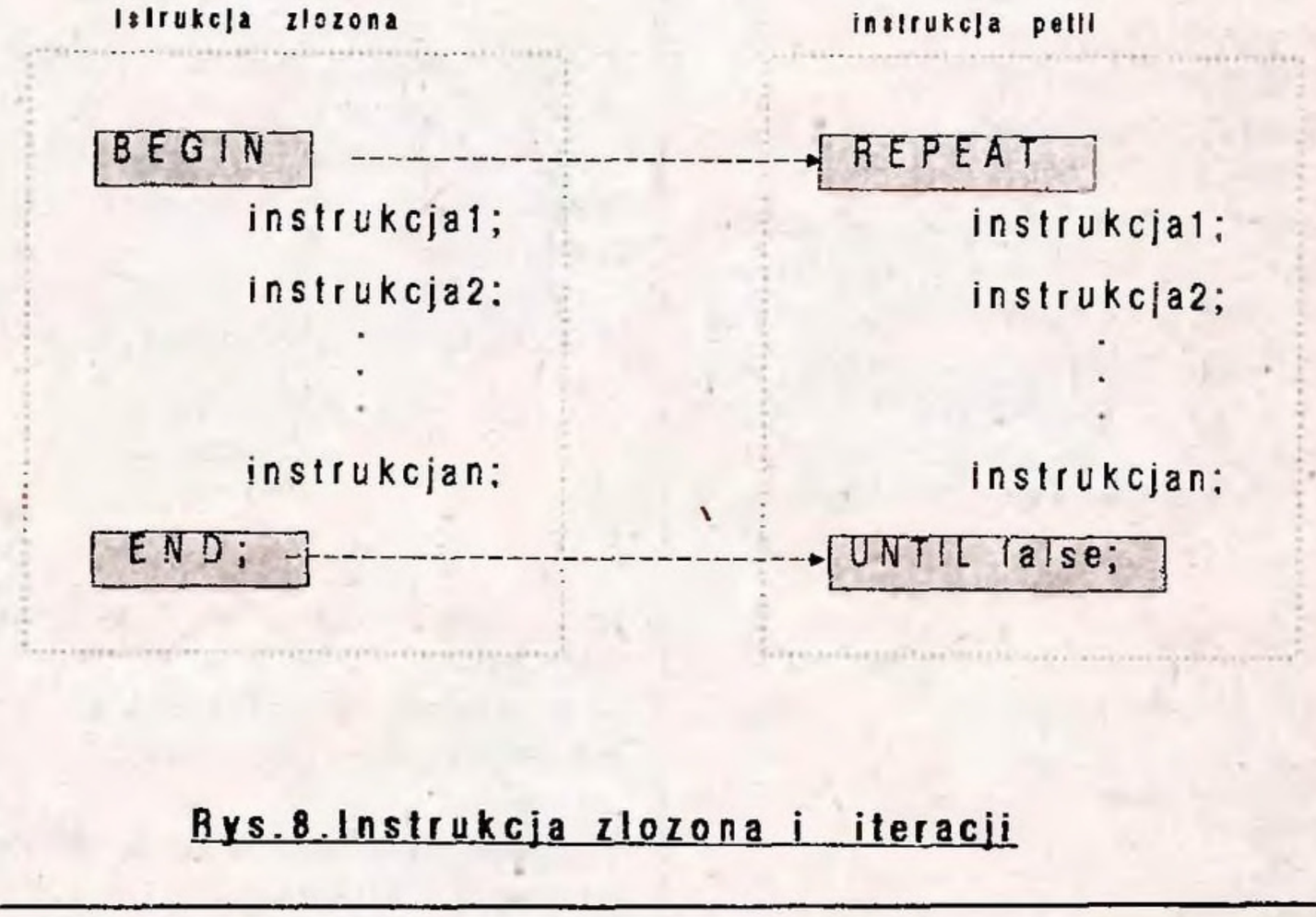
W języku PASCAL istnieje możliwość opisania w bardzo prosty sposób nieskończonego powtarzania czynności. Na takich samych zasadach, jak dla instrukcji złożonej można zbudować powtarzalną sekwencję instrukcji (rys. 8).

W odróżnieniu od instrukcji złożonej, instrukcje występujące pomiędzy słowami kluczowymi **REPEAT—UNTIL** *false* powtarzane są nieskończoną ilość razy.

W przykładzie 19 zamieszczono program naśladujący działanie prostego kalkulatora umożliwiającego wykonanie dodawania, odejmowania, mnożenia i dzielenia dwóch liczb.



Rys.7.Ilustracja do przykladu 18



Rys.8.Instrukcja zlozona i iteracji

Przykład 19

```
PROGRAM prog19;
 VAR
  znak1,znak2:string[2];
  liczba1,liczba2,
  wynik:real;
 BEGIN
  WRITELN;
  REPEAT
   READ(liczba1,znak1,liczba2,znak2);
   IF znak1=' +' THEN
    wynik:=liczba1+liczba2;
   IF znak1=' -' THEN
    wynik:=liczba1-liczba2;
   IF znak1=' *' THEN
    wynik:=liczba1*liczba2;
   IF znak1=' /' THEN
    wynik:=liczba1/liczba2;
   IF ((znak1=' +') OR (znak1=' -') OR
       (znak1=' *') OR (znak1=' /'))
        AND (znak2=' =') THEN
    WRITE(wynik)
   ELSE
    IF ((znak1<>' +') AND (znak1<>' -') AND
      (znak1<>' *') AND (znak1<>' /'))
       OR ((znak2<>' =') AND (znak2<>' K')) THEN
     BEGIN
      WRITELN;
      WRITE('nierozpoznana operacja');
     END;
   WRITELN;
  UNTIL false;
END.
```

Z konstrukcji programu wynika, że akcje naśladowanego kalkulatora będą wykonywane wielokrotnie. Każda akcja polega na wprowadzeniu *liczby, operatora, drugiej liczby* oraz *znaku równości*. Ze względu na właściwości dotychczas poznanych instrukcji wprowadzania danych po każdej liczbie i operatorze konieczne jest wprowadzenie spacji.

Nietrudno zauważyć, że nasz kalkulator nie da się wyłączyć — działania programu nie możemy bowiem przerwać.

Wykorzystana instrukcja **REPEAT — UNTIL** *false* jest szczególnym przypadkiem instrukcji iteracji przyjmującej postać:

REPEAT
instrukcja 1;
instrukcja 2;
.
.
.
instrukcja n;
UNTIL *warunek*

Sekwencja instrukcji zawarta między słowami kluczowymi REPEAT — UNTIL jest powtarzana tak długo, jak długo warunek występujący po UNTIL nie jest spełniony. Spełnienie warunku — sprawdzane każdorazowo po wykonaniu ostatniej instrukcji sekwencji — powoduje zakończenie realizacji instrukcji iteracji. Wynika stąd, że sekwencja instrukcji:

instrukcja 1;
instrukcja 2;
.
.
.
instrukcja n;

zostanie wykonana co najmniej raz.
W szczególnym przypadku warunek może być określony jako wartość logiczna:

— *false* — oznaczająca niespełnienie warunku,

— *true* — oznaczająca spełnienie warunku.

Zatem instrukcja:

REPEAT ... UNTIL *false*

nie zakończy się nigdy, a instrukcja

REPEAT ... UNTIL *true*

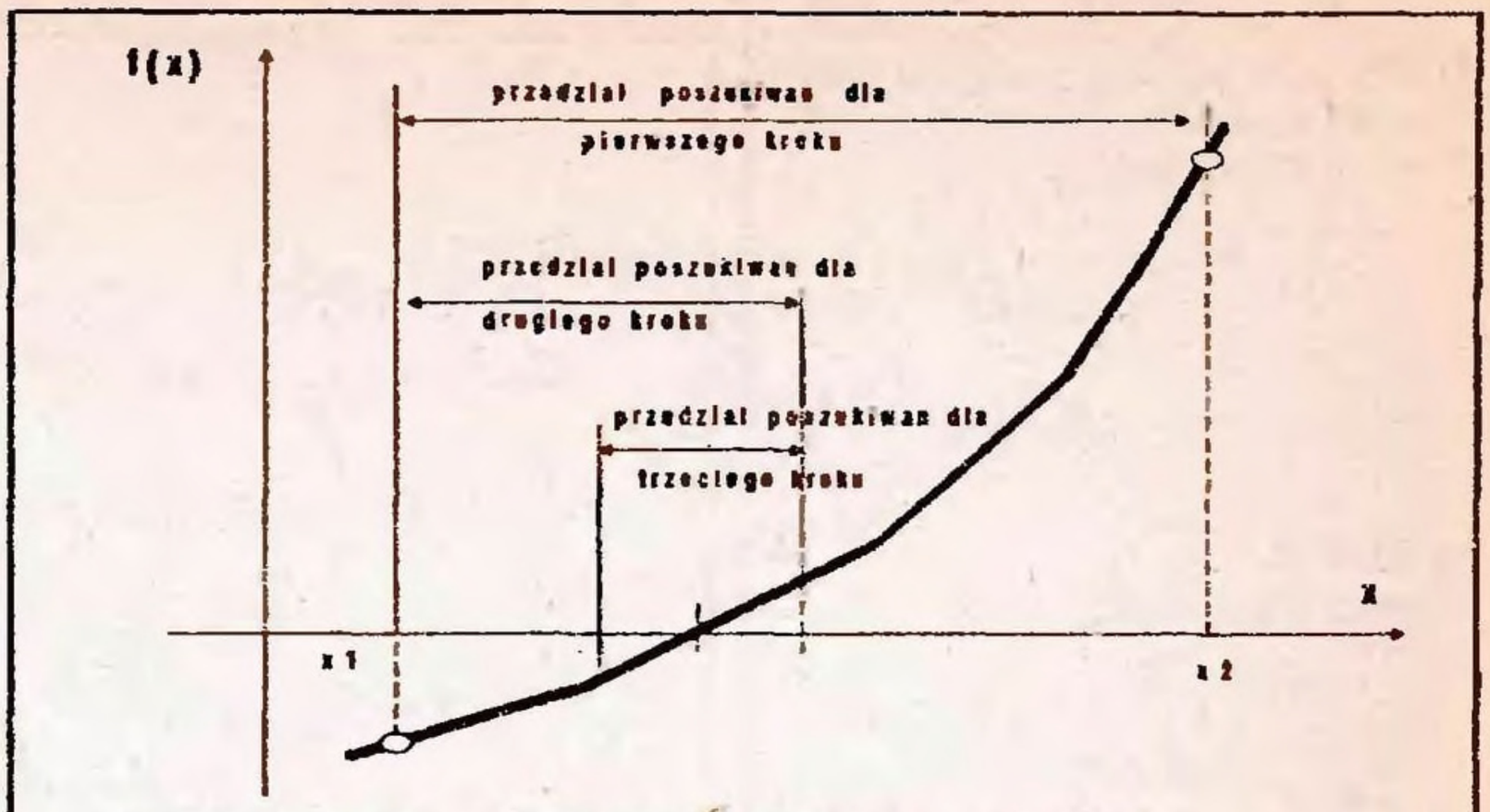
jest równoważna instrukcji złożonej.

Instrukcję iteracji wykorzystano w programie z przykładu 20 rozszerzając program naśladujący działanie kalkulatora o możliwość wyłączenia kalkulatora w przypadku wciśnięcia klawisza *k* zamiast dowolnego operatora.

Przykład 20

```
PROGRAM prog20;
 VAR
  znak1,znak2:string[2];
  liczba1,liczba2,
  wynik:real;
 BEGIN
  WRITELN;
  REPEAT
   READ(liczba1,znak1,liczba2,znak2);
   IF znak1=' +' THEN
    wynik:=liczba1+liczba2;
   IF znak1=' -' THEN
    wynik:=liczba1-liczba2;
   IF znak1=' *' THEN
    wynik:=liczba1*liczba2;
   IF znak1=' /' THEN
    wynik:=liczba1/liczba2;
   IF ((znak1=' +') OR (znak1=' -') OR
       (znak1=' *') OR (znak1=' /'))
        AND (znak2=' =') THEN
    WRITE(wynik)
   ELSE
    IF (znak1<>' +') AND (znak1<>' -') AND
       (znak1<>' *') AND (znak1<>' /') AND
       (znak1<>' k') AND (znak1<>' K') AND
       (znak2<>' k') AND (znak2<>' K') THEN
     BEGIN
      WRITELN;
      WRITE('nierozpoznana operacja');
     END;
   WRITELN;
  UNTIL (znak1=' k') OR (znak1=' K')
     OR (znak2=' k') OR (znak2=' K');
  WRITE('WYLACZENIE KALKULATORA');
 END.
```

Instrukcja iteracji jest bardzo ważnym elementem języka programowania. Dlatego też kolejne cztery przykłady poświęcone są jej wykorzystaniu. Dotyczą one znanych zagadnień matematycznych. Rozważmy problem obliczania wartości szeregu:



Rys.9.Schemat określania przedziału, w którym leży pierwiastek funkcji f(x) metoda połowienia przedziału

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

z zadaną dokładnością. Można go rozwiązać przez dodawanie do siebie kolejno obliczonych wartości wyrazów szeregu. Należy przy tym zauważyć, że:

— wartości kolejnych wyrazów szeregu można wyznaczyć z zależności:

$$a_{n+1} = a_n \frac{x}{n+1}$$

przyjmując, że $a_0 = 1$ (a_n — n-ty wyraz szeregu),

— o dokładności obliczeń decyduje wartość ostatnio obliczonego wyrazu szeregu.

Program obliczania wartości podanego szeregu zamieszczono w przykładzie 21.

Przykład 21

```
PROGRAM prog21;
 VAR
  n,      {numer wyrazu szeregu}
  s,      {wartosc szeregu}
  p,      {wartosc kolejnego wyrazu}
  eps,    {dokladnosc obliczen}
  x:real;
 BEGIN
  s:=0;
  p:=1;
  n:=0;
  READ(x,eps);
  REPEAT
   s:=s+p;
   n:=n+1;
   p:=p*x/(n+1);
  UNTIL abs(p)<=eps;
  WRITELN;
  WRITELN('wartosc szeregu dla x=',x);
  WRITELN('obliczona z dokladnoscia eps=',
          eps);
  WRITE('wynosi s=',s);
 END.
```

Równie popularnym zadaniem jest wyznaczenie miejsca zerowego funkcji f (x) w zadanym przedziale. W praktyce najczęściej kreśli się funkcję

$$y = f(x)$$

i szacuje w przybliżeniu jej miejsce zerowe. Niżej przedstawiono dwie metody wyznaczania miejsca zerowego funkcji w zadanym przedziale wykorzystując twierdzenie:

„jeżeli f(x) jest funkcją ciągłą w przedziale $[x_1, x_2]$ i jeśli $f(x_1)$ i $f(x_2)$ są przeciwnego znaku, to w przedziale $[x_1, x_2]$ istnieje co najmniej jeden pierwiastek rzeczywisty".

Pierwsza metoda, tzw. metoda połowienia przedziałów, pozwala przy zadanym przedziale początkowym $[x_1, x_2]$ określić przedział, w którym znajduje się miejsce zerowe funkcji $f(x)$ z dokładnością *eps*. Postępując zgodnie z tą metodą (jej ideę ilustruje rysunek 9) dzielimy przedział $[x_1, x_2]$ na połowy i w wyznaczonym środku przedziału obliczamy wartość funkcji f_s

$$f_s = f\left(\frac{x_1 + x_2}{2}\right)$$

Następnie badamy czy wartość funkcji w wyznaczonym punkcie zmieniła znak względem $f(x_1)$. Jeżeli tak, to zawężamy przedział do $[x_1, x_s]$, jeżeli nie, to zawężamy do $[x_s, x_2]$

$$X_s = \frac{x_1 + x_2}{2}$$

Po zbadaniu warunków dokładności wyprowadzamy wyniki (jeśli są spełnione) albo powyższe czynności powtarzamy. W przykładzie 22 zamieszczono program określenia przedziału, wykorzystujący metodę połowienia przedziałów, w którym znajduje się miejsce zerowe funkcji:

$$f(x) = -x^3 + 2x^2 + 9$$

z żądaną dokładnością, przy zadanym przedziale początkowym.

Przykład 22

```
PROGRAM prog22;
 VAR
  x1, {dolna granica przedzialu}
  x2, {gorna granica przedzialu}
  xs, {srodek przedzialu}
  f1, {wartosc funkcji w x1}
  fs, {wartosc funkcji w xs}
  eps:real;  {dokladnosc obliczen}
 BEGIN
  WRITELN;
  WRITELN('podaj granice przedzialu',
          ' i dokladnosc obliczen');
  READ(x1,x2,eps);
  REPEAT
   xs:=(x1+x2)/2;
   fs:=-SQR(xs)*xs+2*SQR(xs)+9;
   f1:=-SQR(x1)*x1+2*SQR(x1)+9;
   IF f1*fs<0 THEN
    IF f1*fs<0 THEN
     x2:=xs
    ELSE
     BEGIN
      f1:=fs;
      x1:=xs;
     END;
  UNTIL (ABS(x1-x2)<=eps) OR (f1*fs=0);
  WRITELN;
  WRITELN('znalezione granice przedzialu:');
  WRITELN('x1 = ',x1);
  WRITE('x2 = ',x2);
END.
```

Wykorzystana w programie funkcja **SQR** oblicza kwadrat podanego argumentu (*xs* lub x_1), a *funkcja* **ABS** — wartość bezwzględną argumentu. Idea drugiej metody została zilustrowana na rysunku 10 (jest to tzw. metoda Newtona).

Rozpoczynając od zerowego przybliżenia x_0, kolejne x_1 wyznaczamy przez wykreślenie:
a) linii pionowej $x = x_0$ aż do przecięcia z krzywą $y = f(x)$ — w ten sposób wyznaczony został punkt p_0,
b) stycznej do krzywej $y = f(x)$ w punkcie p_0 aż do przecięcia z osią x w punkcie x_1.

Powtarzając opisane czynności, otrzymujemy ciąg przybliżeń x_0, x_1, x_2,..., którego granica

$$\lim_{i \to \infty} x_i = x$$

jest poszukiwanym miejscem zerowym funkcji *f(x)*. Z rysunku 10 wynika, że tangens kąta nachylenia stycznej poprowadzonej przez punkt p_0 jest równy

$$tg\alpha = \frac{f(x_0)}{x_0 - x_1} = f'(x_0)$$

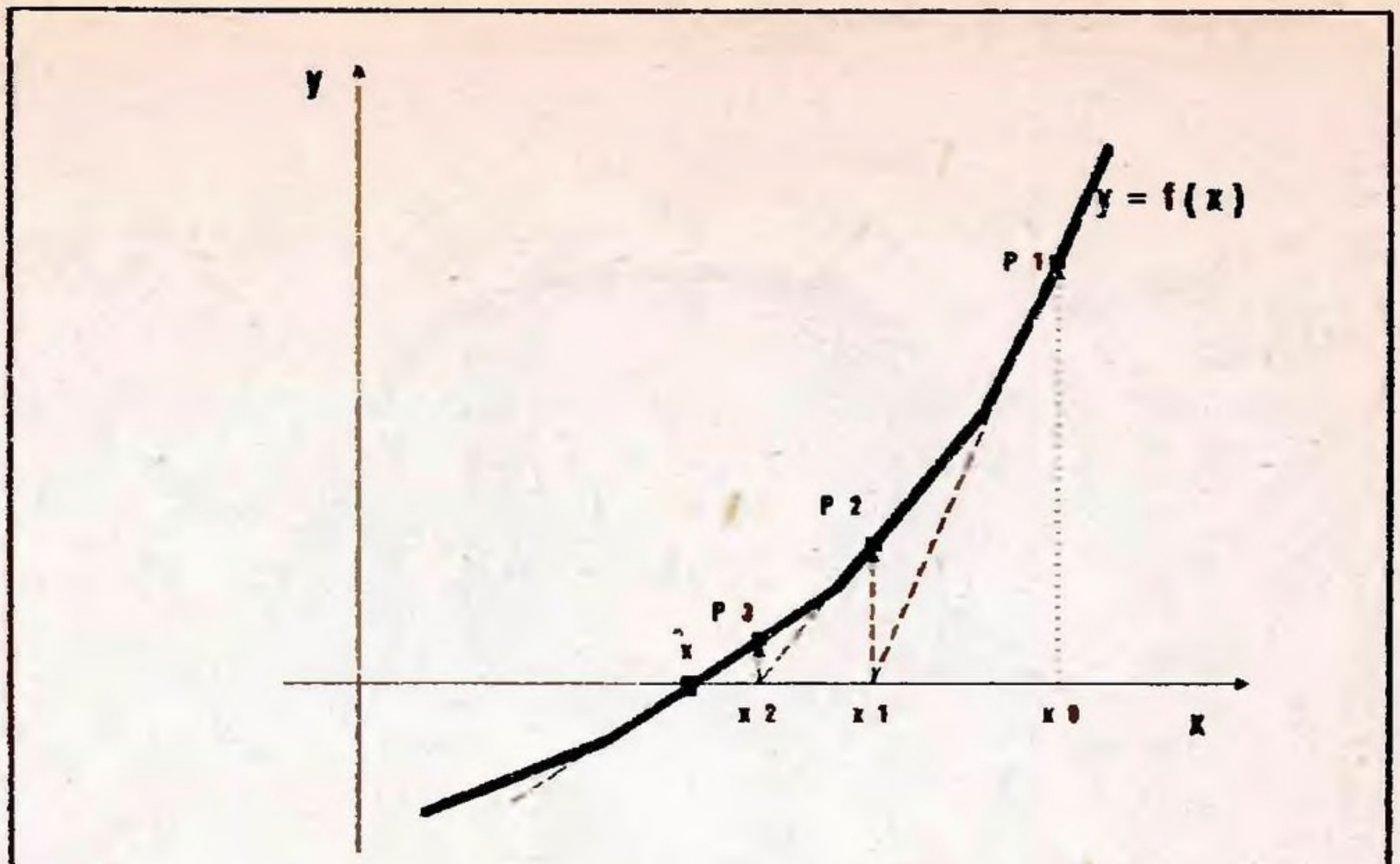
Jeżeli został spełniony warunek $f'(x_0) \neq 0$, to otrzymujemy

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Ogólny wzór na kolejne przybliżenie miejsca zerowego funkcji *f(x)* ma następującą postać:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad dla\ i = 0,1,2...$$

Wzór ten jest słuszny przy założeniu, że $f(x_i) \neq 0$, a zatem w niektórych przypadkach metoda Newtona zawodzi (należy więc wybrać nowy punkt startu).



Rys.10.Graficzna interpretacja metody Newtona dla przypadku gdy f (x)#0

Metodę Newtona wykorzystano do wyznaczenia miejsca zerowego funkcji

$$f(x) = x^4 - 0{,}5\,x^3 - 30$$

z zadaną dokładnością i punktem startu (przykład 23).

Przykład 23

```
PROGRAM prog23;
 VAR
  x,          {punkt aktualny}
  xk,         {punkt kolejny}
  eps,        {zadana dokladnosc obliczen}
  kwx,        {kwadrat x}
  dokl:real;  {biezaca dokladnosc obliczen}
  n,          {maksymalna liczba krokow}
  i:integer;  {licznik krokow}
BEGIN
 WRITELN;
 WRITELN('podaj zerowe przyblizenie, ',
         'dokladnosc obliczen');
 WRITELN('i maksymalna ilosc krokow');
 READ(x,eps,n);
 i:=0;
 REPEAT
  kwx:=x*x;
  xk:=x-(kwx*(kwx-0.5*x)-30)/
      (kwx*(4*x-1.5));
  dokl:=ABS((xk-x)/xk);
  IF dokl>eps THEN
                BEGIN
                 x:=xk;
                 i:=i+1;
                END;
 UNTIL (dokl<=eps) OR (i=n);
 WRITELN;
 WRITELN('miejsce zerowe funkcji x=',x);
 WRITE('wyznaczone w i=',i,' krokach');
END.
```

W przypadku żądania dużej dokładności obliczanie przedstawionego programu może trwać zbyt długo, a dla niektórych przedziałów zmienności funkcji może się w ogóle nie zakończyć (uwaga ta dotyczy wszystkich problemów tego typu).

Dobrze skonstruowany program powinien zawierać górne ograniczenie liczby wykonywanych powtórzeń. Liczba powtórzeń (maksymalna liczba powtórzeń) jest liczbą całkowitą. W języku PASCAL istnieje możliwość deklarowania zmiennych przyjmujących wartości całkowite. W programie **prog23** zostało to wykorzystane do deklaracji licznika kroków *(i)* oraz maksymalnej liczby kroków *(n)*.

Rozważmy teraz problem obliczania całki oznaczonej

$$\int_a^b f(x)\,dx$$

Obliczenie takiej całki jest proste, gdyż *f(x)* jest funkcją elementarną. W przypadkach, gdy nie udaje się łatwo wyznaczyć funkcji pierwotnej, stosujemy metody obliczania przybliżonej wartości całki oznaczonej. Jedną z takich metod jest metoda prostokątów. Ideę metody prostokątów ilustruje rysunek 11.

W metodzie tej przedział całkowania od $a = x_0$ do $b = x_n$ dzieli się na *n* równych części punktami x_1, x_2,..., x_n. Następnie oblicza się wartość funkcji podcałkowej w punktach

$$\frac{x_i - x_{i-1}}{2} \qquad \text{dla } i \in [1,n].$$

Wartość całki jest sumą pól prostokątów utworzonych jak na rysunku 11. Można ją zatem wyrazić wzorem:

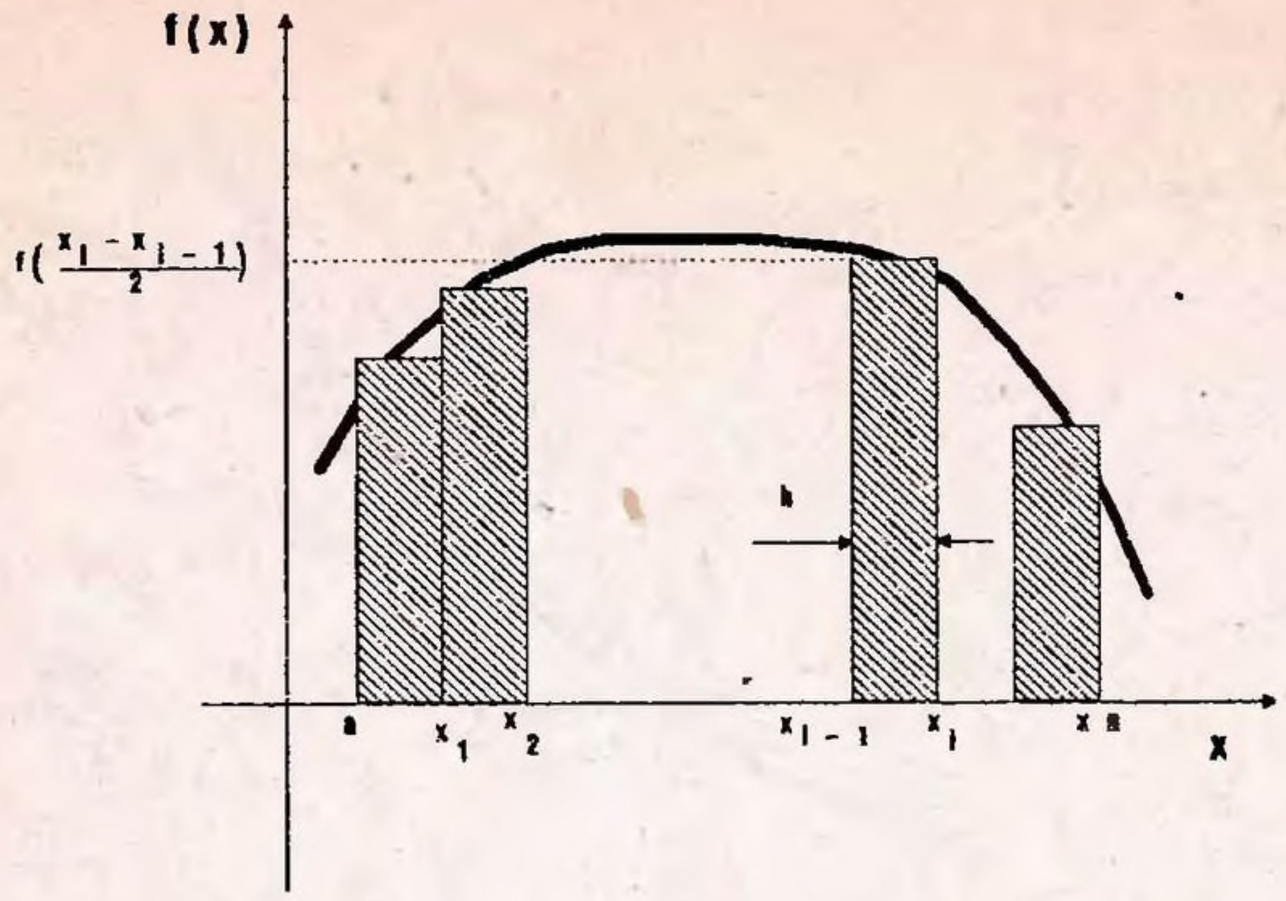
$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} f(x_i)$$

gdzie:
n — liczba podprzedziałów,

$$h = \frac{b-a}{n}$$

$$xj = a + (j - \frac{1}{2})\,h$$

Przykład 24 przedstawia program obliczania całki



Rys.11.Ilustracja metody prostokatow

$$\int_a^b (\ln x + 5x)\,dx$$

metodą prostokątów. Przyjęto w nim, że kolejne przybliżenia całki są obliczane (przy podwojonej każdorazowo liczbie podprzedziałów) dotąd, aż bezwzględna wartość różnicy między dwoma kolejnymi przybliżeniami będzie co najmniej równa wymaganej dokładności obliczeń. Wprowadzono również ograniczenie ilości wykonywanych powtórzeń.

Przykład 24

```
PROGRAM prog24;
 VAR
  a, {dolna granica przedzialu calkowania}
  b, {gorna granica przedzialu calkowania}
  eps:real; {dokladnosc obliczen}
  n:integer; {liczba podprzedzialow}
  h, {szerokosc podprzedzialu}
  w1, {wartosc calki z kroku poprzedniego}
  w2, {aktualna wartosc calki}
  x,      { zmienne }
  i:real; { robocze }
 BEGIN
  WRITELN;
  WRITELN('PODAJ:-dolna granice przedzi',
          'alu calkowania');
  WRITELN(' ':6,'-gorna granice przedzi',
          'alu calkowania');
  WRITELN(' ':6,'-dokladnosc obliczen');
  READ(a,b,eps);
  n:=1;
  h:=b-a;
  x:=a+h/2;
  w2:=LN(x)+5*x;
  REPEAT
   n:=2*n;
   h:=h/2;
   w1:=w2;
   w2:=0;
   x:=a+h/2;
   i:=1;
   REPEAT
    w2:=w2+LN(x)+5*x;
    x:=x+h;
    i:=i+1;
   UNTIL i>n;
   w2:=w2*h;
  UNTIL ABS(w2-w1)<=eps;
  WRITELN;
  WRITELN('wartosc calki dla funkcji ',
          'F(X)=LN(X)+5*X');
  WRITELN('w przedziale ',a,' -',b);
  WRITELN('wyliczona z dokladnoscia ',eps);
  WRITE('wynosi ',w2);
END.
```

Obok instrukcji REPEAT w języku PASCAL występuje druga instrukcja iteracji przyjmująca postać:

WHILE *warunek* **DO** *instrukcja 1*

Powoduje ona, że *instrukcja 1* wykonywana jest tylko wtedy, gdy warunek jest spełniony i tak długo, jak długo jest on spełniony. Oznacza to, że przed każdym wykonaniem *instrukcji 1* następuje sprawdzenie warunku. Niespełnienie warunku powoduje zakończenie powtarzania realizacji *instrukcji 1*. Przykład wykorzystania instrukcji **WHILE** zamieszczono w programie **prog25**, wyznaczającym największy wspólny podzielnik dwóch całkowitych liczb dodatnich.

Przykład 25

```
PROGRAM prog25;
 VAR
  x,y:real;
 BEGIN
  WRITELN('PODAJ x I y');
  READ(x,y);
  WRITELN;
  WRITELN('NAJWIEKSZY WSPOLNY',
          'PODZIELNIK LICZB');
  WRITELN('x =',x:5:0,' y =',y:5:0);
  WHILE x<>y DO
   IF x>y THEN
    x:=x-y
   ELSE
    y:=y-x;
  WRITE('WYNOSI ',x:5:0);
END.
```

Uwaga! Jeśli przed wykonaniem instrukcji WHILE warunek nie jest spełniony, to *instrukcja 1* nie jest w ogóle wykonywana.

ZŁOŻONE TYPY DANYCH

2



Dotychczas poznane instrukcje wyczerpują zbiór najczęściej wykorzystywanych instrukcji języka PASCAL. Jednak o użyteczności języka decydują nie tylko dostępne instrukcje, ale także dostępne dla programisty typy danych. Występujące w przedstawionych wcześniej przykładach typy *string*, *real* i *integer* należą do grupy typów prostych. Oprócz nich w języku PASCAL istnieją również typy strukturalne, umożliwiające opis obiektów, traktowanych jako zbiory wartości typów prostych i strukturalnych. Typy strukturalne są szczególnie przydatne w systemach gromadzenia, przechowywania i przetwarzania dużych ilości informacji. Podstawową cechą takich systemów jest możliwość logicznego grupowania danych. Najprostszą konstrukcją języka PASCAL, pozwalającą na grupowanie danych jest *rekord*. Rekordy umożliwiają opisywanie obiektów, które można scharakteryzować wieloma cechami. Przykładem takiego obiektu może być opis mikrokomputera dostępnego na rynku krajowym. Przyjmując dla uproszczenia, że zawiera on 20-znakową nazwę mikrokomputera, 12-znakową nazwę firmy produkującej mikrokomputer oraz minimalną cenę mikrokomputera (w tys. zł), „rynkowy" mikrokomputer można opisać następująco:

```
RECORD
  nazmik: string [20];
  firma: string [12];
  cenamin: integer
END
```

gdzie:

nazmik, *firma*, *cenamin* — nazwy składowych rekordu zwanych polami, **RECORD**, **END** — para słów kluczowych, oznaczających definicję rekordu.

Definicja rekordu może wystąpić jako element definicji typu zmiennych przyjmującej postać:

TYPE *nazwa typu = określenie typu*

Jeżeli określenie typu będzie definicją powyższego rekordu, a nazwą typu będzie *opmikrok*, to definicja typu przyjmie postać:

TYPE *opmikrok* = **RECORD**
nazmik: string [20];
firma: string [12];
cenamin: integer
END

Definicja ta umożliwia deklarowanie zmiennych np. według formuły

VAR *r1, r2, r3: opmikrok;*

W tym przypadku każda zmienna opisuje inny mikrokomputer. Sposób odwoływania się do pól zmiennej rekordowej przez tzw. nazwy kwalifikowane wyjaśnia przykład 26.

Przykład 26

```
...
TYPE opmikrok=RECORD
                nazmik:string[20];
                firma:string[12];
                cenamin:integer
              END;
    ...
VAR
 rek:opmikrok;
 x:string[20];

    ...
BEGIN
    ...

 rek.nazmik:='AMSTRAD';
 rek.firma:='BEAB';
 rek.cenamin:=750;
    ...

 x:=rek.nazmik;

    ...
```

Po wykonaniu przedstawionej sekwencji instrukcji poszczególnym polom zmiennej *rek* zostaną przypisane wartości. Zmienna *x* (typu takiego jak pole *nazmik*) przyjmie wartość AMSTRAD.

Przedstawiony sposób odwoływania się do pól zmiennej rekordowej może być uciążliwy (zwłaszcza, gdy jest ich dużo lub są to pola typu strukturalnego), gdyż każde pole musi być poprzedzone co najmniej nazwą zmiennej. Wygodniejszy dostęp do pól zmiennej rekordowej umożliwia instrukcja **WITH**, zwana instrukcją wiążącą lub instrukcją dostępu. Pozwala ona na odwoływanie się do pól zmiennej rekordowej bezpośrednio przez nazwy pól, przy jednokrotnym podaniu nazwy zmiennej rekordowej. Wykorzystując instrukcję **WITH**, fragment programu z przykładu 26 można zmodyfikować tak jak podano w przykładzie 27.

Przykład 27

```
    ...
TYPE opmikrok=RECORD
                nazmik:string[20];
                firma:string[12];
                cenamin:integer
              END;
    ...
VAR
 rek:opmikrok;
 x:string[20];

    ...
BEGIN
    ...

WITH rek DO
 BEGIN
   ...
  nazmik:='AMSTRAD';
  firma:='BEAB';
  cenamin:=750;
   ...
  x:=nazmik;
   ...
 END;

    ...
```

W praktyce występuje konieczność gromadzenia danych o identycznej strukturze, których liczba nie jest z góry określona i może się zmieniać w czasie. W języku PASCAL podstawową konstrukcją o takich właściwościach jest plik — sekwencja danych o nieokreślonej długości, przechowywanych w pamięci zewnętrznej (np. dyskowej).

Definicja pliku przyjmuje postać:

TYPE nazwa *pliku* = **FILE OF** nazwa *typu elementu,*

Przykładowa definicja pliku zawierającego rekordy opisujące mikrokomputer jest następująca:

TYPE *zbopmk* = **FILE OF** *opmikrok;*

Z definicją typu nie wiąże się żadna fizyczna reprezentacja. Jest to bowiem informacja o strukturze zmiennych, którym przypisano w deklaracji ten typ. W deklaracji:

VAR *zb1, zb2,: zbopmk;*

powołuje się dwie zmienne *zb1, zb2* reprezentowane jako pliki — zbiory rekordów typu *opmikrok*.

Z plikami związane są operacje:
— tworzenia pliku,
— otwarcia pliku,
— zamknięcia pliku,
— dopisania nowego elementu do pliku,
— odczytu bieżącego elementu z pliku,
— zmiany bieżącego elementu pliku,
— kasowania pliku,
— zmiany nazwy pliku,

Operacje te zostaną omówione w trakcie rozwijania programów wkomponowanych w prosty system obsługi domowej biblioteki obejmujący:
— zakładanie kartoteki (pliku) opisów pozycji,
— dopisywanie nowych opisów pozycji do kartoteki,
— wypisywanie zawartości kartoteki,
— kasowanie wskazanych opisów pozycji z kartoteki.

Przed omówieniem poszczególnych programów niezbędne jest krótkie wyjaśnienie niektórych terminów związanych z operacjami na plikach. W systemie TURBO PASCAL przyjęto, że nazwa pliku nie jest identyfikatorem zbioru danych w pamięci zewnętrznej. Zatem przed wykonaniem jakiejkolwiek operacji na pliku niezbędne jest dowiązanie do nazwy pliku nazwy zbioru w pamięci zewnętrznej z zawartością pliku. Rozwiązanie to realizuje instrukcja

ASSIGN *(nazwa pliku, 'nazwa zbioru')*

gdzie:
nazwa pliku — nazwa zmiennej plikowej (np. zb1),
nazwa zbioru — nazwa zbioru w pamięci zewnętrznej z zawartością pliku (nazwa ta jest łańcuchem znaków o postaci przyjętej dla zbiorów roboczych edytora — patrz „Dodatek").

Po wykonaniu **ASSIGN** wszystkie operacje na pliku są wykonywane na przypisanym mu zbiorze.
Przykładowo:

ASSIGN *(księgozbiór, 'bibldom')*

powoduje dowiązanie zbioru o nazwie *bibldom* do pliku o nazwie *księgozbiór.*

Tworzenie nowego pliku (utworzenie pustego zbioru w pamięci zewnętrznej) realizowane jest za pomocą instrukcji

REWRITE *(nazwa pliku)*

REWRITE tworzy w pamięci zewnętrznej zbiór o nazwie podanej w ASSIGN i udostępnia do zapisu pierwszy element pliku. Jeżeli zbiór o podanej nazwie już istnieje w pamięci zewnętrznej, to jest on kasowany. Po wykonaniu **REWRITE** plik jest pusty, tzn. nie zawiera żadnych elementów.

Po wykorzystaniu pliku należy go zamknąć (przygotować do przechowywania i ponownego wykorzystania) za pomocą instrukcji:

CLOSE *(nazwa pliku)*

Program zakładania pustej kartoteki opisów pozycji biblioteki domowej przedstawiono w przykładzie 28.

Przykład 28

```
PROGRAM prog26;
 TYPE
  pozbibl=RECORD
            autor:string[35];
            tytul:string[50];
            rokwyd,
            nrkat:integer
          END;
  zbbibl=FILE OF pozbibl;
 VAR
  ksiegozbior:zbbibl;
 BEGIN
  ASSIGN(ksiegozbior,'bibldom');
  REWRITE(ksiegozbior);
  CLOSE(ksiegozbior);
 END.
```

W kolejnym przykładzie zamieszczono program dopisywania rekordów na koniec założonej kartoteki.

Wykorzystano w nim instrukcje umożliwiające zmianę bieżącego (zapisywanego lub odczytywanego) elementu pliku. Instrukcja:

SEEK *(nazwa pliku, numer elementu)*

powoduje, że element o wskazanym numerze staje się elementem bieżącym (numer elementu liczony jest od zera). W celu uniknięcia konieczności poszukiwania ostatniego elementu pliku można wykorzystać funkcję

FILESIZE *(nazwa pliku),*

której wartością jest liczba rekordów pliku.
Instrukcja:

SEEK *(nazwa pliku,* **FILESIZE** *(nazwa pliku))*

spowoduje, że elementem bieżącym staje się nowy, pusty element umieszczony na końcu pliku. Zapis do bieżącego elementu pliku realizuje instrukcja

WRITE *(nazwa pliku, nazwa zmiennej)*

przy czym *nazwa zmiennej* określa zmienną, której wartość jest zapisywana do bieżącego elementu pliku. Typ zmiennej musi być zgodny z typem elementów pliku.

Otwarcie istniejącego pliku jest wykonywane przez instrukcję.

RESET *(nazwa pliku)*

Należy przypomnieć, że instrukcja **REWRITE** nie może tu być wykorzystywana, bowiem niszczy ona dotychczasową zawartość pliku. Instrukcja **RESET** może być używana tylko w odniesieniu do już istniejących plików. W wyniku wykonania **RESET** elementem bieżącym staje się pierwszy element pliku.

W szczególnym przypadku jest on pusty.

Przykład 29

```
PROGRAM prog27;
 TYPE
  pozbibl=RECORD
           autor:string[35];
           tytul:string[50];
           rokwyd,
           nrkat:integer
          END;
 zbbibl=FILE OF pozbibl;
VAR
 ksiegozbior:zbbibl;
 rek:pozbibl;
 znkon:string[1];
BEGIN
 ASSIGN(ksiegozbior,'bibldom');
 RESET(ksiegozbior);
 SEEK(ksiegozbior,FILESIZE(ksiegozbior));
 WRITELN;
 REPEAT
  WITH rek DO
   BEGIN
    WRITELN('podaj nazwisko i imie autora',
            ' (do 35 znakow)');
    READLN(autor);
    WRITELN('podaj tytuł ksiazki',
            ' (do 50 znakow)');
    READLN(tytul);
    WRITELN('podaj rok wydania');
    READLN(rokwyd);
    WRITELN('nadaj numer katalogowy');
    READLN(nrkat);
   END;
  WRITE(ksiegozbior,rek);
  WRITELN('czy wprowadzasz nastepna ',
          'pozycje? (T/N)');
  READLN(znkon);
 UNTIL (znkon='N') OR (znkon='n');
 CLOSE(ksiegozbior);
END.
```

W uzupełnieniu do przykładu 29 należy dodać, że program **prog27** może być wykorzystywany wielokrotnie bez względu na zawartość kartoteki. Przy jednokrotnym uruchomieniu programu można dopisywać do kartoteki dowolną liczbę opisów pozycji.

Istotnym elementem systemu gromadzenia informacji jest możliwość przeglądania zawartości kartotek. Program przeglądania naszej kartoteki zamieszczono w przykładzie 30. Wykorzystano w nim instrukcję:

READ *(nazwa pliku, nazwa zmiennej)*

powodującą przypisanie zmiennej *nazwa zmiennej* wartości bieżącego elementu. Podobnie, jak dla instrukcji **WRITE**, powinien być spełniony warunek zgodności typów. Ponadto wykorzystano funkcję logiczną:

EOF *(nazwa pliku)*

której wartością jest *true*, jeżeli bieżący element jest ostatnim elementem pliku, *false* — w przypadku przeciwnym.

Przykład 30

```
PROGRAM prog29;
 TYPE
  pozbibl=RECORD
           autor:string[35];
           tytul:string[50];
           rokwyd,
           nrkat:integer
          END;
 zbbibl=FILE OF pozbibl;
VAR
 ksiegozbior:zbbibl;
 rek:pozbibl;
 lpom:string[1];
BEGIN
 ASSIGN(ksiegozbior,'bibldom');
 RESET(ksiegozbior);
 CLRSCR;
 WHILE NOT EOF(ksiegozbior) DO
  BEGIN
   READ(ksiegozbior,rek);
   WITH rek DO
    BEGIN
     WRITELN('AUTOR: ',autor);
     WRITELN('TYTUL: ',tytul);
     WRITELN('ROK WYDANIA: ',rokwyd);
     WRITELN('NUMER KATALOGOWY: ',nrkat);
    END;
   WRITELN('========================',
           '========================');
   WRITELN;
   WRITELN(' ':20,'- VERTE -');
   READ(lpom);
   CLRSCR;
  END;
 CLOSE(ksiegozbior);
END.
```

Ponieważ zawartość całego pliku może się nie zmieścić na ekranie, program **prog28** został tak skonstruowany, że wyświetlenie kolejnej pozycji uwarunkowane jest wciśnięciem klawisza **ENTER**.

W kolejnym przykładzie zamieszczono program umożliwiający kasowanie wskazanych opisów pozycji kartoteki.

W programie zastosowano bardzo prostą metodę usuwania opisów. Usunięcie jednego opisu powoduje przepisanie wszystkich pozostałych opisów do pomocniczego zbioru *zbpom* (rysunek 12).

Następnie kasowany jest zbiór *bibldom*, a zbiór *zbpom* przyjmuje nazwę *bibldom*. W programie wykorzystano dwie nowe instrukcje:

ERASE *(nazwa pliku)*

powodującą skasowanie zbioru związanego z plikiem oraz:

RENAME *(nazwa pliku, 'nazwa zbioru')*

zmieniającą nazwę zbioru związanego z plikiem na nazwę *nazwa zbioru*. Obie powyższe instrukcje mogą być wykonywane tylko na plikach, które zostały uprzednio zamknięte za pomocą instrukcji **CLOSE**.

Przykład 31

```
PROGRAM prog29;
 TYPE
  pozbibl=RECORD
           autor:string[35];
           tytul:string[50];
           rokwyd,
           nrkat:integer
          END;
 zbbibl=FILE OF pozbibl;
 VAR
  ksiegozbior,
  zbpom:zbbibl;
  rek:pozbibl;
  idpozkas:string[55];
  znkon:string[1];
BEGIN
 REPEAT
  ASSIGN(ksiegozbior,'bibldom');
  ASSIGN(zbpom,'pom.dta');
  RESET(ksiegozbior);
  REWRITE(zbpom);
  WRITELN;
  WRITELN('podaj tytul kasowanej pozycji');
  READLN(idpozkas);
  WHILE NOT EOF(ksiegozbior) DO
   BEGIN
    READ(ksiegozbior,rek);
    IF rek.tytul<>idpozkas THEN
       WRITE(zbpom,rek);
   END;
  CLOSE(ksiegozbior);
  CLOSE(zbpom);
  ERASE(ksiegozbior);
  RENAME(zbpom,'bibldom');
  WRITELN;
  WRITELN('czy kasujesz nastepna pozycje?',
          ' (T/N)');
  READ(znkon);
 UNTIL (znkon='N') OR (znkon='n');
END.
```

Pliki umożliwiają gromadzenie i przechowywanie danych w pamięciach zewnętrznych. Jest to ich niewątpliwą zaletą. Do wad plików można zaliczyć długi czas dostępu do ich elementów (ze względu na wymagany kontakt z pamięcią zewnętrzną).

Alternatywną formą deklarowania zbiorów danych w języku PASCAL są zmienne tablicowe. Nie mają one swojej reprezentacji w pamięciach zewnętrznych, przechowywane są w pamięci komputera i „giną" wraz z

zakończeniem programu. Liczba ich elementów jest z góry określona. Dane przechowywane są w postaci tablic jedno-, dwu- lub wielowymiarowych.

Jednowymiarowa tablica, to ciąg ponumerowanych elementów, których wartości są tego samego typu. Dwuwymiarowa tablica, to ciąg wierszy lub kolumn, z których każda liczy tyle samo elementów tego samego typu. Rozmiary dwuwymiarowej tablicy, to liczba wierszy i liczba elementów w wierszu czyli liczba kolumn. Za pomocą jednowymiarowych tablic opisujemy zwykle takie obiekty jak wektory, natomiast za pomocą tablic dwuwymiarowych — macierze. Ponieważ nie przewidujemy przedstawiania problemów wymagających tablic o wymiarach większych niż 2, poprzestaniemy na omówieniu tablic co najwyżej dwuwymiarowych.

Naturalnym sposobem opisu tablic jest notacja:

a) **ARRAY** [*1.. n*] **OF** *integer*
dla opisu tablicy jednowymiarowej o n elementach typu integer,

b) **ARRAY** [*1.. m, 1.. n*] **OF** *real*
dla opisu tablicy dwuwymiarowej o rozmiarach m x n i elementach typu real.

W przykładzie 32 zamieszczono prosty program nadawania wartości elementom jednowymiarowej tablicy a.

Przykład 32

```
PROGRAM prog30;
 VAR
  a:ARRAY[1..6] OF integer;
  i:integer;
 BEGIN
  WRITELN;
  i:=1;
  WHILE i<=6 DO
   BEGIN
    a[i]:=i;
    WRITELN(a[i]);
    i:=i+1;
   END;
 END.
```

Działania na elementach tablicy przeprowadza się tak jak na zmiennych typu zgodnego z typem elementów tablicy. Każdy element tablicy jest identyfikowany jednoznacznie przez nazwę tablicy oraz tzw. indeks, określający położenie elementów w tablicy, np.:

a [*i*] — wskazywany przez wartość zmiennej *i* element jednowymiarowej tablicy *a*.

a [*i — 1*] — wskazywany przez wartość wyrażenia *(i — 1)*, element jednowymiarowej tablicy *a*.

W przypadku, gdy indeks jest wyrażeniem, wartość wyrażenia jest obliczana przed udostępnieniem elementu tablicy. Należy podkreślić, że bieżąca wartość indeksu musi mieścić się w przedziale wartości dopuszczalnych dla tego indeksu (w programie **prog30** — [1,10]).

W przykładzie 32 do opisu powtarzających się czynności wykorzystano znaną już instrukcję **WHILE**. Nietrudno zauważyć, że w tym przypadku stosowanie tej instrukcji nie jest zbyt wygodne. Konieczne jest bowiem ustalenie liczby powtórzeń operacji wykonanych w instrukcji **WHILE** oraz określenie warunku kończącego jej wykonanie. Warunek powinien być spełniony wtedy, gdy osiągnie się wymaganą liczbę powtórzeń.

Nieprecyzyjne określenie warunku prowadzi do niepoprawnej liczby wykonanych powtórzeń (najczęściej o 1 mniejszej lub o 1 większej od wymaganej).

Do opisu czynności powtarzanych określoną ilość razy wygodniejsza jest instrukcja o postaci:

FOR *I*: = e_1 **TO** e_2 **DO** *instrukcja 1,*

gdzie:

I — zmienna kontrolna (licznik powtórzeń),

e_1 — wyrażenie określające początkową wartość zmiennej kontrolnej,

e_2 — wyrażenie określające końcową wartość zmiennej kontrolnej.

Zapis ten oznacza, że *instrukcja 1* wykonywana jest dla kolejnych wartości zmiennej kontrolnej *I* od wartości początkowej e_1 z krokiem 1 do wartości końcowej e_2. Wartość zmiennej kontrolnej jest zmieniana automatycznie, zatem zmiana taka nie jest dopuszczalna w ramach *instrukcji 1*. Zmienna kontrolna może być natomiast wykorzystywana jako argument w wyrażeniach, jeżeli takie występują w *instrukcji 1*. Przyjmuje się, że po wykonaniu instrukcji **FOR** wartość zmiennej kontrolnej jest nieokreślona.

W przykładzie 33 zamieszczono zmodyfikowany program **prog31**, w którym zamiast instrukcji **WHILE** wykorzystano instrukcję **FOR**.

Przykład 33

```
PROGRAM prog31;
 VAR
  a:ARRAY[1..6] OF integer;
  i:integer;
 BEGIN
  WRITELN;
  FOR i:=1 TO 6 DO
   BEGIN
    a[i]:=i;
    WRITELN(a[i]);
   END;
 END.
```

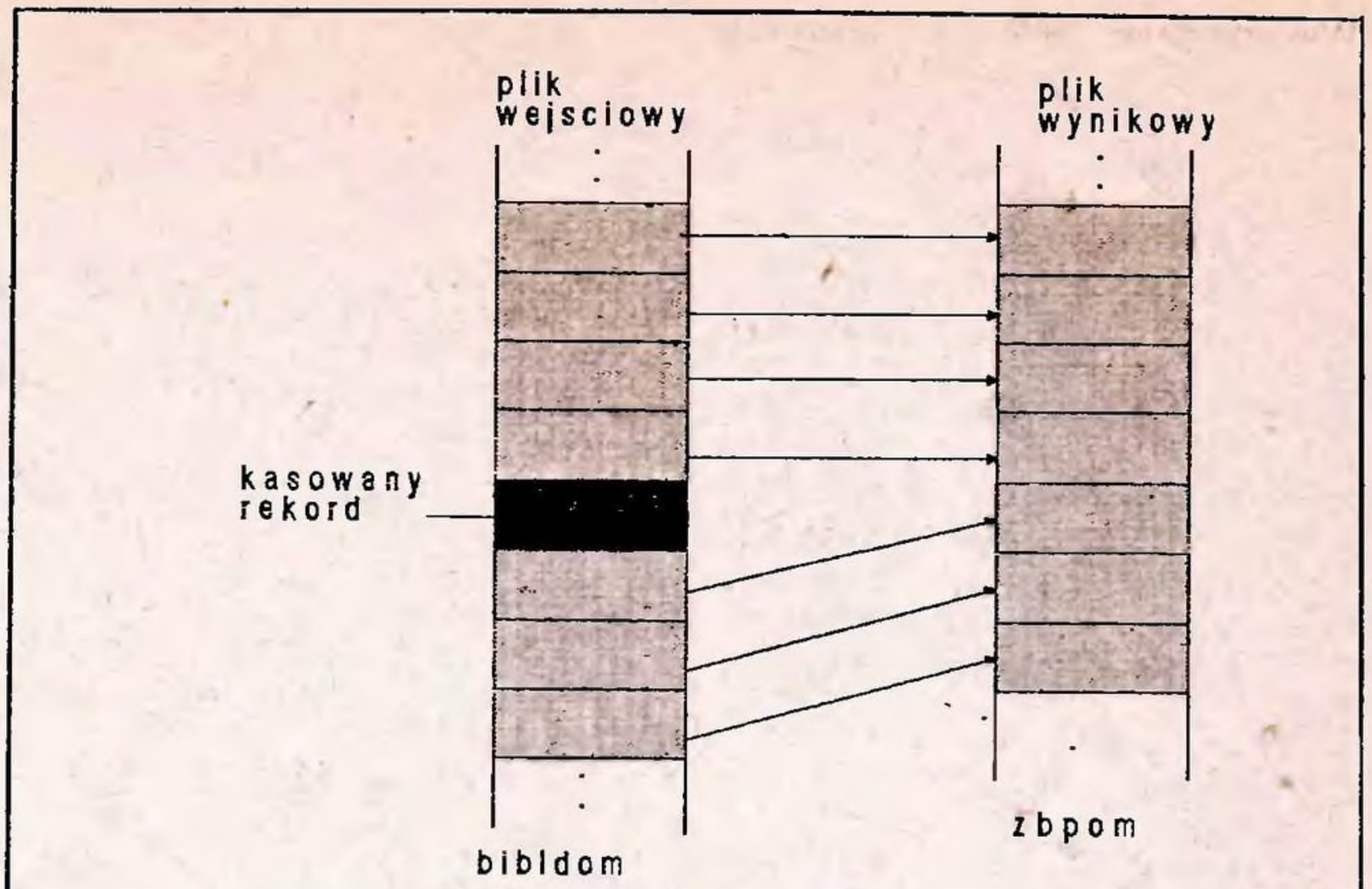
Aby zamknąć listę instrukcji iteracyjnych języka PASCAL opiszemy drugą postać instrukcji FOR. Jest ona następująca:

FOR *I*: = e_1 **DOWNTO** e_2 **DO** *instrukcja 1*

gdzie:

I, e_1, e_2, *instrukcja 1* — mają identyczne znaczenie, jak w poprzednio omówionej postaci instrukcji FOR.

Zapis ten oznacza, że *instrukcja 1* wykonywana jest dla kolejnych wartości zmiennej kontrolnej *I* od wartości początkowej e_1 do wartości końcowej e_2, przy czym wartość zmiennej kontrolnej jest zmniejszona o 1 po każdym wykonaniu *instrukcji 1*. Przedstawiona postać instrukcji występuje w programie porządkowania sześciu zadanych liczb całkowitych w sposób rosnący (przykład 34) metodą tzw. sortowania bąbelkowego. Metoda ta opiera się na zasadzie porównywania i zamiany par sąsiadujących



Rys.12. Ilustracja metody kasowania rekordu z pliku

ze sobą liczb, do chwili ustawienia ich w kolejności od najmniejszej do największej. Wynika stąd, że warunkiem uporządkowania jest jednoczesne udostępnienie wszystkich sześciu liczb. Jeżeli potraktujemy zadane liczby jako elementy „pionowej" tablicy jednowymiarowej, to przy pewnej dozie wyobraźni możemy je uważać za znajdujące się w naczyniu z wodą bąbelki o „wadze" proporcjonalnej do ich wielkości. Rezultatem każdego przejścia przez tablicę będzie wypchnięcie bąbelka na poziom odpowiadający jego wadze. Przebieg sortowania bąbelkowego ilustruje tabela 1.

TABELA 1

Ustawienie początkowe	Pierwsze przejście	Drugie przejście	Trzecie przejście	Czwarte przejście	Piąte przejście
574	→ 8	8	8	8	8
303	574	→ 23	23	23	23
34	303	574	→ 34	34	34
125	34	303	574	→ 125	125
8 →	125	34 →	303	574	→ 303
23	23 →	125	125 →	303 →	574

Przykład 34

```
PROGRAM prog32;
 VAR
  liczby:ARRAY[1..6] OF integer;
  x,i,j:integer;
 BEGIN
  FOR i:=1 TO 6 DO
   READ(liczby[i]);
  WRITELN;
  WRITELN('wprowadzone liczby:');
  FOR i:=1 TO 6 DO
   WRITE(liczby[i]:5,' ':3);
  FOR i:=2 TO 6 DO
   FOR j:=6 DOWNTO i DO
    IF liczby[j-1]>liczby[j] THEN
     BEGIN
      x:=liczby[j-1];
      liczby[j-1]:=liczby[j];
      liczby[j]:=x;
     END;
  WRITELN;
  WRITELN('liczby uporzadkowane:');
  FOR i:=1 TO 6 DO
   WRITE(liczby[i]:5,' ':3);
 END.
```

Kolejne dwa przykłady dotyczą działań na macierzach. Przykład 35 przedstawia program dodawania dwóch macierzy o ustalonych wymiarach. Przypomnijmy, że sumą dwóch macierzy **A** i **B** o tych samych wymiarach jest trzecia macierz **C**, której elementy wyznaczane są wg zależności:

$$c_{ij} = a_{ij} + b_{ij}$$

gdzie:

i — indeks wiersza,
j — indeks kolumny.

W programie, w deklaracji zmiennych tablicowych *(a, b, c)*, wykorzystano typ *macierz* opisujący macierze o wymiarach 3x5 i elementach typu *real*. Wprowadzenie elementów macierzy *a* i *b* następuje wierszami.

Przykład 35

```
PROGRAM prog33;
 TYPE
  macierz=ARRAY[1..3,1..5] OF real;
 VAR
  a,b,          {macierze wejsciowe}
  c:macierz;    {macierz wyjsciowa}
  i,j:integer; {zmienne pomocnicze}
 BEGIN
  WRITELN;
  WRITELN('wprowadz elementy macierzy A');
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     READ(a[i,j]);
    READLN;
   END;
  WRITELN('wprowadz elementy macierzy B');
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     READ(b[i,j]);
    READLN;
   END;
  FOR i:=1 TO 3 DO
   FOR j:=1 TO 5 DO
    c[i,j]:=a[i,j]+b[i,j];
  WRITELN;
  WRITELN('dane wejsciowe:');
  WRITELN(' ':10,'macierz A');
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     WRITE(' ':3,a[i,j]:10);
    WRITELN;
   END;
  WRITELN(' ':10,'macierz B');
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     WRITE(' ':3,b[i,j]:10);
    WRITELN;
   END;
  WRITELN('wynik:');
  WRITELN(' ':10,'macierz C=A+B');
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     WRITE(' ':3,c[i,j]:10);
    WRITELN;
   END;
 END.
```

Zastanówmy się, co należałoby zmienić w programach z przykładów 34 i 35 w przypadku innych wymiarów tablic.

Nietrudno zauważyć, że zmiana wymiarów „rozprzestrzeni" się na cały program. Należy bowiem zmodyfikować również wszystkie instrukcje programu, w których występują wartości graniczne wymiarów tablic. Zmiana wymiarów tablicy *liczby* z przykładu 34 wymaga wprowadzenia zmian w sześciu wierszach programu, a zmiana wymiarów tablicy *macierz* z przykładu 35 — aż w trzynastu wierszach.

Wad tych jest pozbawiony program mnożenia macierzy przedstawiony w przykładzie 36. Przypomnijmy, że iloczyn **A**x**B** dwóch macierzy **A** i **B** jest taką macierzą **C**, w której element wiersza *i* i kolumny *j* jest sumą iloczynów elementów wiersza *i* macierzy **A** przez odpowiednie elementy kolumny *j* macierzy **B**. Liczba kolumn macierzy **A** musi być przy tym równa liczbie wierszy macierzy **B**. Jeżeli więc macierz **A** jest stopnia $p \times q$ i macierz **B** — stopnia $q \times r$, to macierz **C** jest stopnia $p \times r$.

$$\mathbf{C}_{pxr} = \mathbf{A}_{pxq} \cdot \mathbf{B}_{qxr}$$

gdzie:

$$c_{ij} = \sum_{k=1}^{q} a_{ik} \cdot b_{kj}$$

Wymiary tablic zostały określone przez nazwy stałych *(dw, gw, dk, gk)*, zdefiniowanych po słowie kluczowym **CONST**. Ponieważ w instrukcjach programu użyto również zdefiniowanych stałych, nie jest on „wrażliwy" na zmiany wymiarów tablic. Ewentualna zmiana wymiarów tablic wymaga jedynie przedefiniowania odpowiednich stałych.

Przykład 36

```
PROGRAM prog34;
 CONST
  dw=1; {dolna wartosc indeksu wierszy}
  gw=7; {gorna wartosc indeksu wierszy}
  dk=1; {dolna wartosc indeksu kolumn}
  gk=7; {gorna wartosc indeksu kolumn}
 TYPE
  tablica=ARRAY[dw..gw,dk..gk] OF real;
 VAR
  p,          {liczba wierszy macierzy A}
  q1,         {liczba kolumn macierzy A}
  q2,         {liczba wierszy macierzy B}
  r:integer; {liczba kolumn macierzy B}
  a,b,        {macierze wejsciowe}
  c:tablica;  {macierz wynikowa}
  i,j,k:integer; { zmienne }
  s:real;        { robocze }
BEGIN
 WRITELN;
 WRITELN('podaj wymiary macierzy A');
 READLN(p,q1);
 IF (p>gw) OR (q1>gk) THEN
  BEGIN
   WRITELN;
   WRITELN('niepoprawne wymiary macierzy A');
  END
 ELSE
  BEGIN
   WRITELN('podaj wymiary macierzy B');
   READLN(q2,r);
   IF (q2>gw) OR (r>gk) THEN
    BEGIN
     WRITELN;
     WRITELN('niepoprawne wymiary macierzy B');
    END
   ELSE
    IF q1<>q2 THEN
     BEGIN
      WRITELN;
      WRITELN('niezgodnosc wymiarow macie',
              'rzy A i B');
     END
    ELSE
     BEGIN
      WRITELN('wprowadz elementy macierzy A');
      FOR i:=dw TO p DO
       BEGIN
        FOR j:=dk TO q1 DO
         READ(a[i,j]);
        READLN;
       END;
      WRITELN('wprowadz elementy macierzy B');
      FOR i:=dw TO q2 DO
       BEGIN
        FOR j:=dk TO r DO
         READ(b[i,j]);
        READLN;
       END;
```

```
FOR i:=dw TO q2 DO
 FOR j:=dk TO r DO
  BEGIN
   s:=0;
   FOR k:=dk TO q1 DO
    s:=s+a[i,k]*b[k,j];
   c[i,j]:=s;
  END;
WRITELN;
WRITELN(' ':5,'macierz A');
FOR i:=dw TO p DO
 BEGIN
  FOR j:=dk TO q1 DO
   WRITE(a[i,j]:10,' ');
  WRITELN;
 END;
WRITELN(' ':5,'macierz B');
FOR i:=dw TO q2 DO
 BEGIN
  FOR j:=dk TO r DO
   WRITE(b[i,j]:10,' ');
  WRITELN;
 END;
WRITELN(' ':5,'macierz C=A*B');
FOR i:=dw TO p DO
 BEGIN
  FOR j:=dk TO r DO
   WRITE(c[i,j]:10,' ');
  WRITELN;
 END;
END.
```

Analizując program **prog34** można zauważyć, że aktualnie wymagane wymiary tablic są wczytywane w czasie wykonywania programu. Pozwala to na zmianę wymiarów tablic bez konieczności modyfikacji tekstu programu. Oczywiście zakres zmian jest ograniczony zdefiniowanymi wymiarami tablic. Po wczytaniu bieżących wartości wymiarów tablic należy sprawdzić czy mieszczą się one w przedziale wartości dopuszczalnych.

W dotychczasowych przykładach zaprezentowano dwa sposoby opisu zbiorów danych: w postaci plików oraz tablic. W przykładzie 37 zamieszczono program sortowania zawartości pliku znaną już metodą sortowania bąbelkowego. Ponieważ metoda ta wymaga jednoczesnego udostępnienia wszystkich sortowanych elementów, przed sortowaniem wszystkie elementy pliku zostały umieszczone w tablicy. Zwiększa się w ten sposób szybkość sortowania, ale dla większych plików metody tej zastosować nie można (ze względu na ograniczoną pojemność pamięci komputera). W tych przypadkach trzeba stosować inne metody, których ze względu na ich złożoność nie prezentujemy.

Przykład 37

```
PROGRAM prog35;
 CONST
  n=100;
 TYPE
  pozbibl=RECORD
            autor:string[35];
            tytul:string[50];
            rokwyd,
            nrkat:integer
           END;
  zbbibl=FILE OF pozbibl;
  zbpao=ARRAY[1..100] OF pozbibl;
 VAR
  ksiegozbior:zbbibl;
  tabpom:zbpao;
  rek:pozbibl;
  i,j,ilop:integer;
 BEGIN
  ASSIGN(ksiegozbior,'bibldom');
  RESET(ksiegozbior);
  ilop:=0;
  WHILE NOT EOF(ksiegozbior) DO
   BEGIN
    ilop:=ilop+1;
    READ(ksiegozbior,tabpom[ilop]);
   END;
  FOR i:=2 TO ilop DO
   FOR j:=ilop DOWNTO i DO
    IF tabpom[j-1].autor >
       tabpom[j].autor THEN
     BEGIN
      rek:=tabpom[j-1];
      tabpom[j-1]:=tabpom[j];
      tabpom[j]:=rek;
     END;
  RESET(ksiegozbior);
  FOR i:=1 TO ilop DO
   WRITE(ksiegozbior,tabpom[i]);
  CLOSE(ksiegozbior);
 END.
```

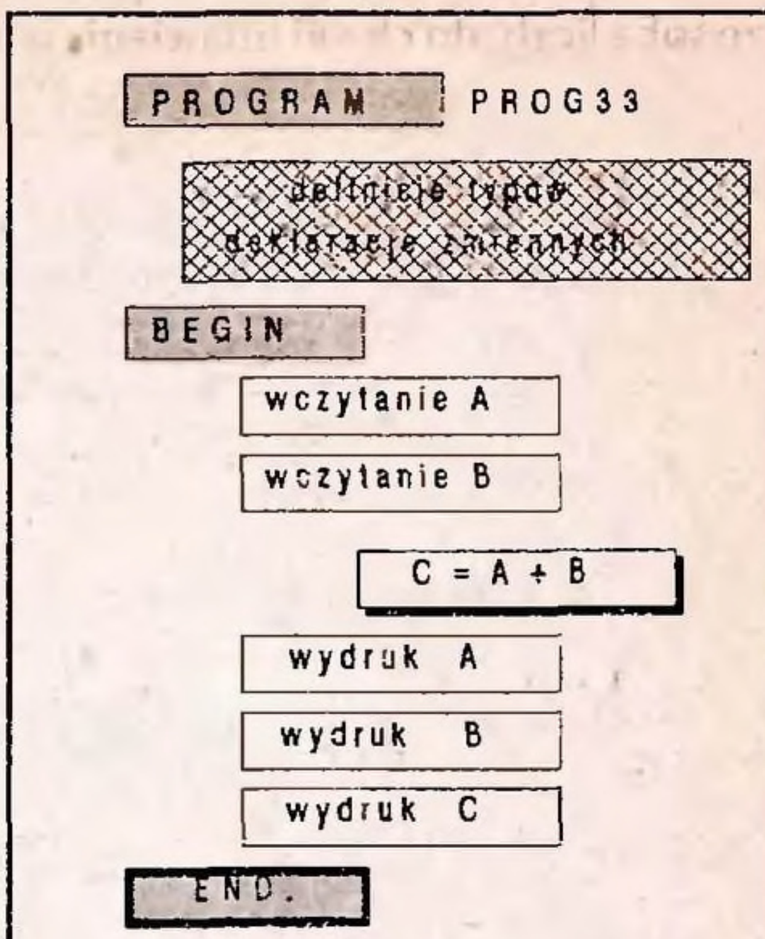
Należy zwrócić uwagę na typ elementów tablicy, w której przeprowadza się sortowanie — elementami tymi są rekordy odpowiadające elementom pliku. Po przesortowaniu elementy tablicy są zapisywane do tego samego pliku.

PROCEDURY I FUNKCJE

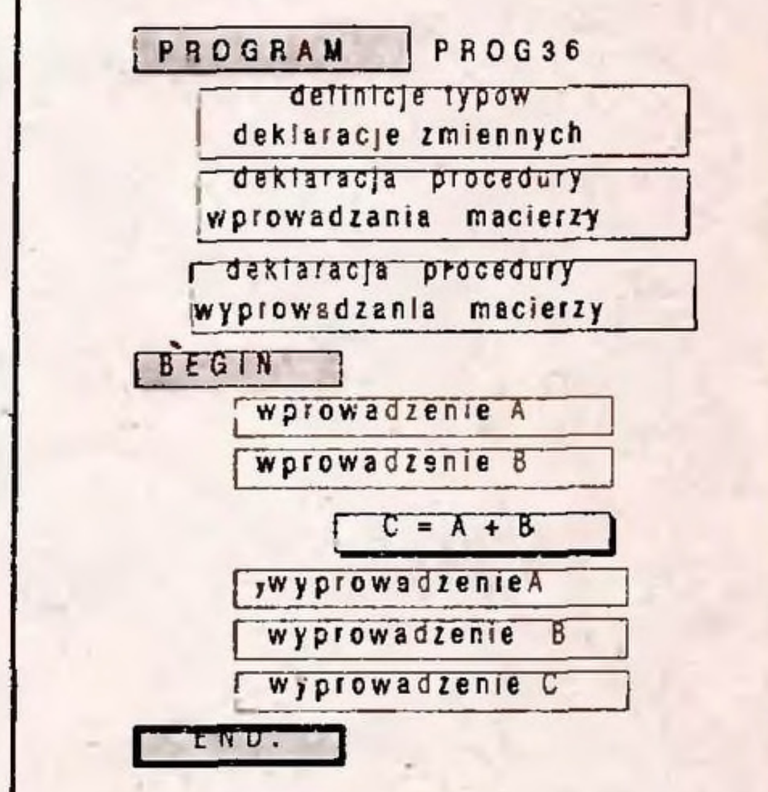
3



Programy przedstawione w przykładach 35 i 36 należą do najdłuższych spośród dotychczas prezentowanych. Istotny wpływ na ich długość mają powtarzające się sekwencje instrukcji wprowadzania i wyprowadzania elementów macierzy. Strukturę programu z przykładu 35 zilustrowano rysunkiem 13.



Rys.13.Struktura programu PROG33



Rys.14.Struktura programu z przykładu 35

Sytuacja taka występuje w wielu programach. Często bowiem w różnych miejscach programu występuje potrzeba wykonywania niemal identycznych sekwencji instrukcji. Dla programisty wygodne jest więc wyodrębnienie takich sekwencji i traktowanie ich jako swego rodzaju wzorców, zwanych w językach programowania **procedurami.** Wzorcem takim może być np. sekwencja wprowadzania elementów macierzy:

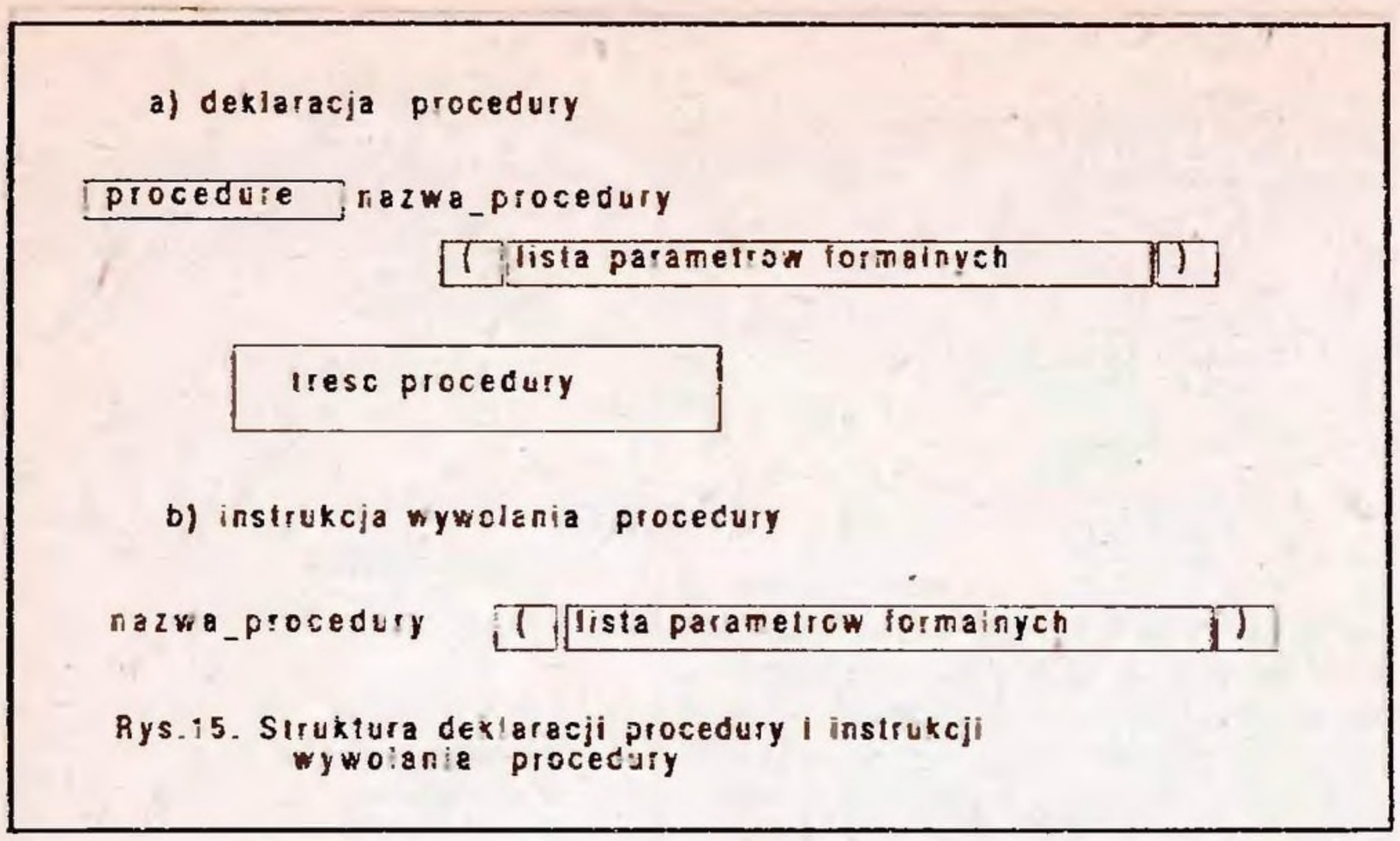
```
FOR j: = 1 TO 3 DO
 BEGIN;
  FOR j:=1 TO 5 DO
   READ (x [i, j])];
READLN;
END;
```

W miejscu wystąpienia nazwy macierzy wprowadzono w niej znak *x* symbolizujący tę nazwę. Symbol *x* może być traktowany tak jak traktowany jest argument funkcji, gdyż wynik wykonania procedury jest uwarunkowany nazwą konkretnej macierzy.

W języku PASCAL istnieje możliwość deklarowania procedur (wzorców sekwencji instrukcji wraz z listą argumentów, zwanych parametrami formalnymi) oraz możliwość wykonywania procedur z konkretnymi wartościami parametrów (zwanych parametrami aktualnymi), drogą wywołania specjalnej instrukcji zwanej instrukcją procedury. Uwzględniając powyższe możliwości struktura programu z rysunku 13 przyjmie postać, jak na rysunku 14.

Z powyższego rysunku wynika, że logika programu nie uległa zmianie. Dodatkowymi jego elementami są deklaracje procedur, zawierające powtarzalne sekwencje instrukcji. Strukturę deklaracji i wywołania procedury zilustrowano rysunkiem 15.

Z rysunku wynika, że treść procedury opatrzona jest nazwą *(nazwa procedury)* wraz z listą parametrów formalnych, a instrukcja procedury zawiera jedynie nazwę procedury (a nie treść) oraz listę parametrów aktualnych. W odniesieniu do wcześniej przedstawionej sekwencji wprowadzania elementów macierzy, deklara-



Rys.15. Struktura deklaracji procedury i instrukcji wywołanie procedury

cja odpowiadającej procedury przyjmie postać:

```
PROCEDURE czytmac
                    (VAR x : macierz);
VAR i, j: integer;
BEGIN
FOR i: = 1 TO 3 DO
 BEGIN
  FOR j: = 1 TO 5 DO
   READ (x [i, j]);
 READLN;
 END
```

a odpowiadająca jej instrukcja wywołania przyjmie postać:

czytmac(a)

dla wprowadzania elementów macierzy a i postać:

czytmac(b)

dla wprowadzania elementów macierzy *b*.

W liście parametrów formalnych procedury *czytmac* zadeklarowano obiekt x typu *macierz* z programu **prog33**.

Należy dodać, że:

a) jeżeli parametr formalny jest poprzedzony słowem kluczowym **VAR**, to wskazuje ono, że wartość parametru aktualnego może ulec zmianie w wyniku wykonania procedury (brak słowa **VAR** zapewnia niezmienność wartości parametru aktualnego, bez względu na treść procedury);
b) treść procedury ma budowę analogiczną do budowy programu i obejmuje:
   — definicje stałych,
   — definicje typów,
   — deklaracje zmiennych,
   — deklaracje procedur,
   — listę instrukcji (zawartą pomiędzy **BEGIN** i **END**);
c) deklaracja procedury zakończona jest znakiem „;", a instrukcja wywołania procedury traktowana jest tak jak każda instrukcja.

Program **prog36** (dodawania macierzy z uwzględnieniem procedur) zamieszczono w przykładzie 38.

Przykład 38

```
PROGRAM prog36;
 TYPE
  macierz=ARRAY[1..3,1..5] OF real;
 VAR
  a,b,           {macierze wejsciowe}
  c:macierz;     {macierz wyjsciowa}
  i,j:integer; {zmienne pomocnicze}
PROCEDURE czytmac(VAR x:macierz);
 VAR
  i,j:integer;
 BEGIN
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     READ(x[i,j]);
    READLN;
   END;
 END;
PROCEDURE piszmac(VAR x:macierz);
 VAR
  i,j:integer;
 BEGIN
  FOR i:=1 TO 3 DO
   BEGIN
    FOR j:=1 TO 5 DO
     WRITE(' ':3,x[i,j]:10);
    WRITELN;
   END;
 END;
BEGIN
 WRITELN;
 WRITELN('wprowadz elementy macierzy A');
 czytmac(a);
 WRITELN('wprowadz elementy macierzy B');
 czytmac(b);
 FOR i:=1 TO 3 DO
  FOR j:=1 TO 5 DO
   c[i,j]:=a[i,j]+b[i,j];
 WRITELN;
 WRITELN('dane wejsciowe:');
 WRITELN(' ':10,'macierz A');
 piszmac(a);
 WRITELN(' ':10,'macierz B');
 piszmac(b);
 WRITELN('wynik:');
 WRITELN(' ':10,'macierz C=A+B');
 piszmac(c);
END.
```

Z przykładu 38 wynika, że celem wyodrębnienia procedur jest skracanie tekstu programu. W rzeczywistości jednak podstawowym celem wyodrębniania procedur jest dekompozycja programu na fragmenty, z których każdy stanowi logicznie zamkniętą całość. Ma to szczególne znaczenie w przypadku projektowania dużych programów, ale efekty konsekwentnego stosowania podziału programu na jednostki logiczne dają się zauważyć w każdym programie. Uzyskuje się bowiem większą przejrzystość programu i znacznie zmniejsza możliwość popełnienia błędu. Ilustracją sposobu dekompozycji jest przykład 39. Zamieszczono w nim program obliczania sumy, różnicy, iloczynu i ilorazu dwóch liczb zespolonych, w którym każde działanie zostało potraktowane jako oddzielna procedura.

Przykład 39

```
PROGRAM prog37;
 TYPE
  liczbazesp=RECORD
               re,      {czesc rzeczywista}
               im:real  {czesc urojona    }
             END;
 VAR
  l1,l2,wyn:liczbazesp;
PROCEDURE sumazesp(x,y:liczbazesp;
                   VAR z:liczbazesp);
 BEGIN
  z.re:=x.re+y.re;
  z.im:=x.im+y.im;
 END;
PROCEDURE roznicazesp(x,y:liczbazesp;
                      VAR z:liczbazesp);
 BEGIN
  z.re:=x.re-y.re;
  z.im:=x.im-y.im;
 END;
PROCEDURE iloczynzesp(x,y:liczbazesp;
                      VAR z:liczbazesp);
 BEGIN
  z.re:=x.rey.re-x.imy.im;
  z.im:=x.rey.im+x.imy.re;
 END;
PROCEDURE ilorazzesp(x,y:liczbazesp;
                     VAR z:liczbazesp);
 VAR
  pom:real;
 BEGIN
  pom:=y.rey.re+y.imy.im;
  z.re:=(x.rey.re+x.imy.im)/pom;
  z.im:=(x.imy.re-x.rey.im)/pom;
 END;
 PROCEDURE wypiszzesp(l:liczbazesp);
  BEGIN
   WRITE(' ':5,l.re);
   IF l.im>=0 THEN WRITE(' +');
   writeln(l.im,'i');
  END;
BEGIN
 WRITELN;
 WRITELN('PODAJ CZESC RZECZYWISTA ',
         'PIERWSZEJ LICZBY');
 READLN(l1.re);
 WRITELN('PODAJ CZESC UROJONA ',
         'PIERWSZEJ LICZBY');
 READLN(l1.im);
 WRITELN('PODAJ CZESC RZECZYWISTA ',
         'DRUGIEJ LICZBY');
 READLN(l2.re);
 WRITELN('PODAJ CZESC UROJONA ',
         'DRUGIEJ LICZBY');
 READ(l2.im);
 WRITELN;
 WRITELN('suma podanych liczb:');
 sumazesp(l1,l2,wyn);
 wypiszzesp(wyn);
 WRITELN('roznica podanych liczb:');
 roznicazesp(l1,l2,wyn);
 wypiszzesp(wyn);
 WRITELN('iloczyn podanych liczb:');
 iloczynzesp(l1,l2,wyn);
 wypiszzesp(wyn);
 IF (l2.re=0) AND (l2.im=0) THEN
  WRITE('iloraz podanych liczb nie ',
        'moze byc wyznaczony')
 ELSE
  BEGIN
   WRITELN('iloraz podanych liczb:');
   ilorazzesp(l1,l2,wyn);
   wypiszzesp(wyn);
  END;
END.
```

W przypadku bardziej złożonych programów ich teksty mimo stosowania procedur są bardzo obszerne. TURBO PASCAL umożliwia zwiększenie przejrzystości poprzez włączenie we wskazane miejsca fragmentów tekstu programu wcześniej

przygotowanego. Są one wskazywane w specjalnej dyrektywie kompilatora o strukturze:

{$I nazwa zbioru}

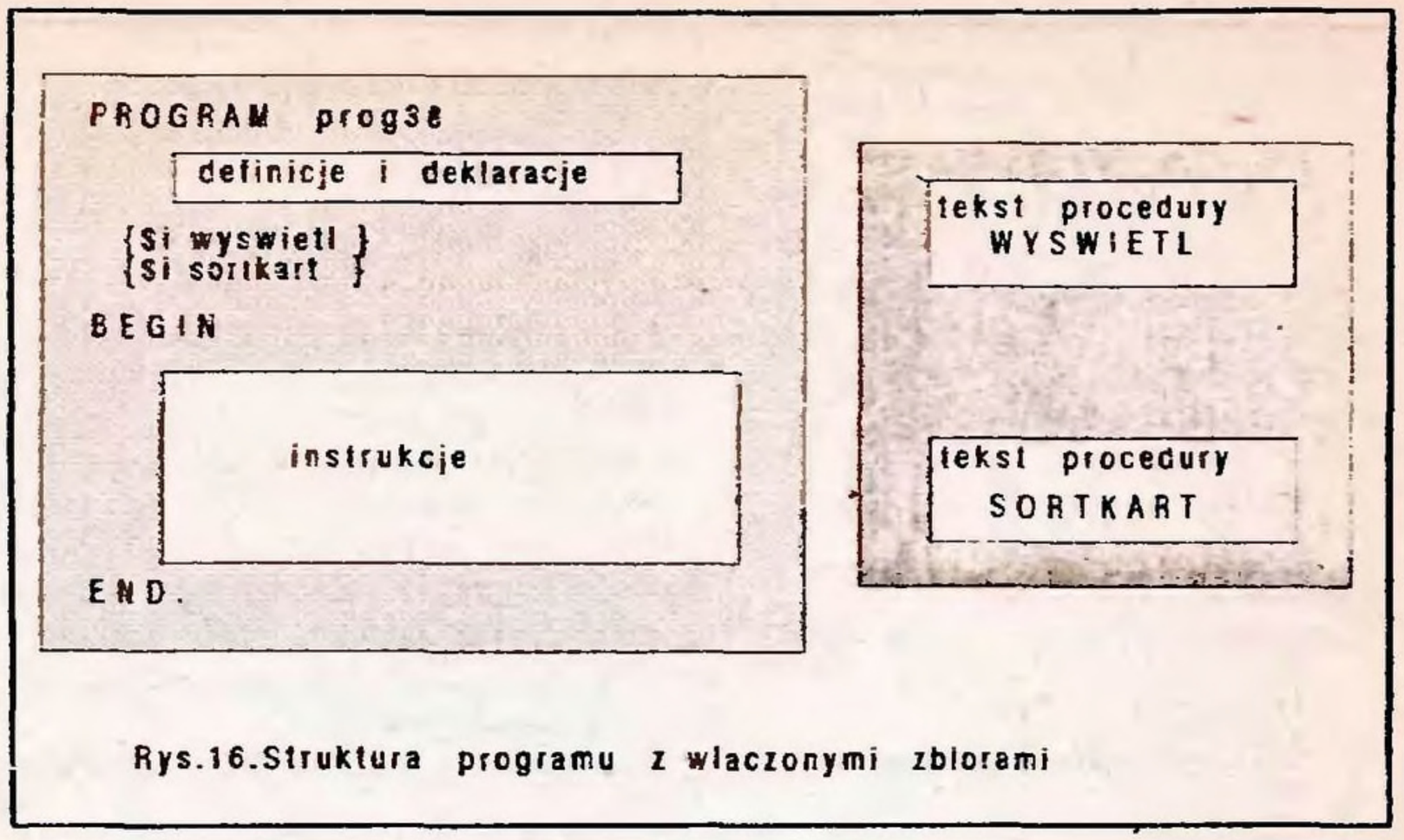
Dyrektywa ta zawiera nazwę zbioru w pamięci dyskowej, w której przechowywany jest odpowiedni fragment tekstu programu. Sposób użytkowania tej przydatnej dyrektywy został zilustrowany przykładem 40. Zamieszczono w nim program wyprowadzający zawartość założonej wcześniej kartoteki (przykład 29) przed i po uporządkowaniu jej elementów wg klucza, którym jest nazwisko autora. Przyjęto, że teksty dwóch wyodrębnionych procedur: wyprowadzania zawartości kartoteki (przykład 41) i porządkowania elementów kartoteki (przykład 42), znajdują się w oddzielnych zbiorach o nazwach: **WYSWIETL.PAS** i **SORTKART.PAS.**

Przykład 40

```
PROGRAM prog38;
 TYPE
  pozbibl=RECORD
            autor:string[35];
            tytul:string[50];
            rokwyd,
            nrkat:integer
          END;
  zbbibl=FILE OF pozbibl;
 VAR
  ksiegozbior:zbbibl;
  nazwazb:string[12];
  zn:string[1];
  {$i wyswietl.pas}
  {$i sortkart.pas}
BEGIN
 WRITELN;
 WRITELN('PODAJ NAZWE ZBIORU');
 READ(nazwazb);
 ASSIGN(ksiegozbior,nazwazb);
 wyswkart(ksiegozbior);
 WRITELN('CZY WYPROWADZASZ KARTOTEKE',
         ' UPORZADKOWANA (T/N)?');
 READ(zn);
 IF (zn='t') OR (zn='T') THEN
  BEGIN
   sortkart(ksiegozbior);
   wyswkart(ksiegozbior);
  END;
END.
```

Przykład 41

```
PROCEDURE wyswkart(VAR zb:zbbibl);
 VAR
  i:integer;
  rek:pozbibl;
  lpom:string[1];
 BEGIN
  RESET(zb);
  CLRSCR;
  WHILE NOT EOF(zb) DO
   BEGIN
    READ(zb,rek);
    WITH rek DO
     BEGIN
      WRITELN('AUTOR: ',autor);
      WRITELN('TYTUL: ',tytul);
      WRITELN('ROK WYDANIA: ',rokwyd);
      WRITELN('NUMER KATALOGOWY: ',nrkat);
     END;
    WRITELN('========================',
            '========================');
    WRITELN;
    WRITELN(' ':20,'- WERTE -');
    READ(lpom);
    CLRSCR;
   END;
  CLOSE(zb);
 END;
```



Rys.16.Struktura programu z wlaczonymi zbiorami

Przykład 42

```
PROCEDURE sortkart(VAR zb:zbbibl);
 CONST
  n=100;
 TYPE
  zbpac=ARRAY[1..100] OF pozbibl;
 VAR
  tabpom:zbpac;
  rek:pozbibl;
  i,j,ilop:integer;
 BEGIN
  RESET(zb);
  ilop:=0;
  WHILE NOT EOF(zb) DO
   BEGIN
    ilop:=ilop+1;
    READ(zb,tabpom[ilop]);
   END;
  FOR i:=2 TO ilop DO
   FOR j:=ilop DOWNTO i DO
    IF tabpom[j-1].autor >
       tabpom[j].autor THEN
     BEGIN
      rek:=tabpom[j-1];
      tabpom[j-1]:=tabpom[j];
      tabpom[j]:=rek;
     END;
  RESET(zb);
  FOR i:=1 TO ilop DO
   WRITE(zb,tabpom[i]);
  CLOSE(zb);
 END;
```

Zbiory **WYSWIETL.PAS** oraz **SORTKART.PAS** tworzone są tak, jak gdyby stanowiły oddzielne programy. Dodajmy, że tekst całego programu (tzn. z włączonymi zbiorami) musi mieć strukturę akceptowaną przez kompilator (rys. 16).

W przytoczonych wyżej przykładach logicznie wyodrębnionym fragmentem programu była procedura traktowana jako sekwencja instrukcji. Jej wywołanie jest więc instrukcją. Inne właściwości mają procedury funkcyjne, zwane dalej **funkcjami. Funkcja,** podobnie jak procedura, jest wyodrębnionym fragmentem tekstu programu. Wynikiem jej wywołania jest wartość (typu określonego w nagłówku funkcji) obliczona w treści funkcji. Wywołanie funkcji nie jest więc instrukcją, lecz elementem wyrażenia lub warunku.

Struktura deklaracji funkcji jest następująca:

FUNCTION *nazwa funkcji* (*lista parametrów formalnych*) : *typ wyniku* ; *treść funkcji*

W treści funkcji musi wystąpić co najmniej jedna instrukcja przypisania o strukturze:

nazwa *funkcji* : = *wyrażenie*

która wartość przypisywanego wyrażenia pozwala traktować jako wynik funkcji. Jeżeli taka instrukcja nie wystąpi w treści funkcji lub nie zostanie wykonana, to wartość funkcji będzie nieokreślona.

Wywołanie funkcji ma taką samą postać, jak wywołanie procedury.

W przykładzie 43 zamieszczono program obliczania ilości *k*-elementowych kombinacji w zbiorze *n*-elementowym wg wzoru:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Wydzielono tam funkcję obliczającą wartość silni wskazanej liczby.

Przykład 43

```
PROGRAM prog39;
 VAR
  n,k:integer;
  ilkomb:real;
 FUNCTION silnia(x:integer):real;
  VAR
   i:integer;
   pom:real;
  BEGIN
   IF x=0 THEN silnia:=1
   ELSE
    BEGIN
     pom:=1;
     FOR i:=1 TO x DO
      pom:=pom*i;
     silnia:=pom;
    END;
  END;
BEGIN
 WRITELN;
 WRITELN('podaj N i K');
 READ(n,k);
 ilkomb:=silnia(n)/(silnia(k)*silnia(n-k));
 WRITELN;
 WRITELN('ilosc kombinacji z ',n,' po ',k);
 WRITE('wynosi ',ilkomb);
END.
```

W przykładzie 43 liczby naturalne deklarowane są jako rzeczywiste. Wynika to z bardzo ograniczonego (do 32767) zakresu liczb typu *integer*.

W kolejnym przykładzie zamieszczono program obliczania wartości wyrażenia:

$$x^n \cdot sinh\,(x)$$

W programie wyodrębniono funkcję podnoszenia liczby *x* do potęgi *n*(n — liczba całkowita) oraz funkcję obliczania *sinh (x)* wg zależności.

$$sinh\,(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots$$

Przykład 44

```
PROGRAM prog40;
 VAR
  n:integer;
  x,wynik:real;
 FUNCTION potega(x:real;n:integer):real;
  VAR
   i:integer;
   pom:real;
  BEGIN
   pom:=1;
   IF n<>0 THEN
    FOR i:=1 TO ABS(n) DO
     pom:=pom*x;
   IF n<0 THEN potega:=1/pom
   ELSE potega:=pom;
  END;
 FUNCTION sh(x:real):real;
  VAR
   n,s,p:real;
  BEGIN
   s:=0;
   p:=x;
   n:=0;
   REPEAT
    s:=s+p;
    n:=n+1;
    p:=p*x*x/(2*n*(2*n+1));
   UNTIL abs(p)<=0.001;
   sh:=s;
  END;
BEGIN
 WRITELN;
 WRITELN('podaj X i N');
 READ(x,n);
 wynik:=potega(x,n)*sh(x);
 WRITELN;
 WRITELN('obliczona wartosc wyrazenia');
 WRITELN('dla X = ',x,' i N = ',n);
 WRITE('wynosi ',wynik);
END.
```

Poznany zakres właściwości języka PASCAL umożliwia rozwiązywanie zadań poprzez sekwencyjne wykonywanie instrukcji lub poprzez wielokrotne wykonywanie tych samych instrukcji aż do spełnienia pożądanych warunków.

Konstruując programy główną uwagę należy zwrócić na poprawność logiczną oraz warunki zakończenia programu (brak nieskończonych pętli).

Istnieje szereg problemów, które dają się opisać metodami rekursji. W metodach tych zakłada się wykorzystanie tej samej funkcji (procedury) z innymi argumentami do obliczenia jej wartości. Przykładem może tu być definicja funkcji silni:

$$n! = \begin{cases} 1 & dla\ n = 0 \\ n\,(n-1)! & dla\ n > 0 \end{cases}$$

w której do obliczenia jej wartości dla argumentu *n* wykorzystuje się wartość funkcji dla argumentu n — *1*. Definicję taką można zapisać wykorzystując deklarację funkcji rekurencyjnej języka PASCAL (przykład 45).

Przykład 45

```
FUNCTION silnia(n:integer):integer;
 BEGIN
  IF n=0 THEN
     silnia:=1
  ELSE
   silnia:=n*silnia(n-1);
 END;
```

Szczególną własnością takich funkcji jest umieszczenie wywołania funkcji w jej treści. Podobną regułę można zastosować w stosunku do procedur rekurencyjnych. W przykładzie 46 zamieszczono procedurę porządkowania elementów jednowymiarowej tablicy liczb całkowitych. Porządkowanie odbywa się wg następującej reguły.

1) wyszukuje się element najmniejszy,

2) zamienia się miejscami element najmniejszy z pierwszym (bieżącym) elementem tablicy,

3) kroki 1,2 i 3 są powtarzane z pominięciem pierwszego (bieżącego) elementu tablicy.

W kroku 3 powtarza się więc czynności 1,2 i 3, zawężając pole poszukiwań elementu minimalnego.

Przykład 46

```
PROCEDURE sortuj(m,n:integer;
                 VAR tab:ciag);
 PROCEDURE zmienmin(k,l:integer);
  VAR
   elmin,nrelmin,
   i,pom:integer;
  BEGIN
   nrelmin:=k;
   elmin:=tab[k];
   FOR i:=k+1 TO l DO
    IF tab[i]<elmin THEN
      BEGIN
       nrelmin:=i;
       elmin:=tab[i];
      END;
   pom:=tab[nrelmin];
   tab[nrelmin]:=tab[k];
   tab[k]:=pom;
 END;
BEGIN
 IF m<>n THEN
  BEGIN
   zmienmin(m,n);
   sortuj(m+1,n,tab);
  END;
END;
```

Podobnie jak dla funkcji, w deklaracji procedury rekurencyjnej występuje jej wywołanie.

Programowanie z wykorzystaniem rekurencji jest zadaniem trudnym. Niżej zamieszczono dwa przykłady ilustrujące zastosowanie rekurencji do rozwiązywania zagadnień z dziedziny kombinatoryki.

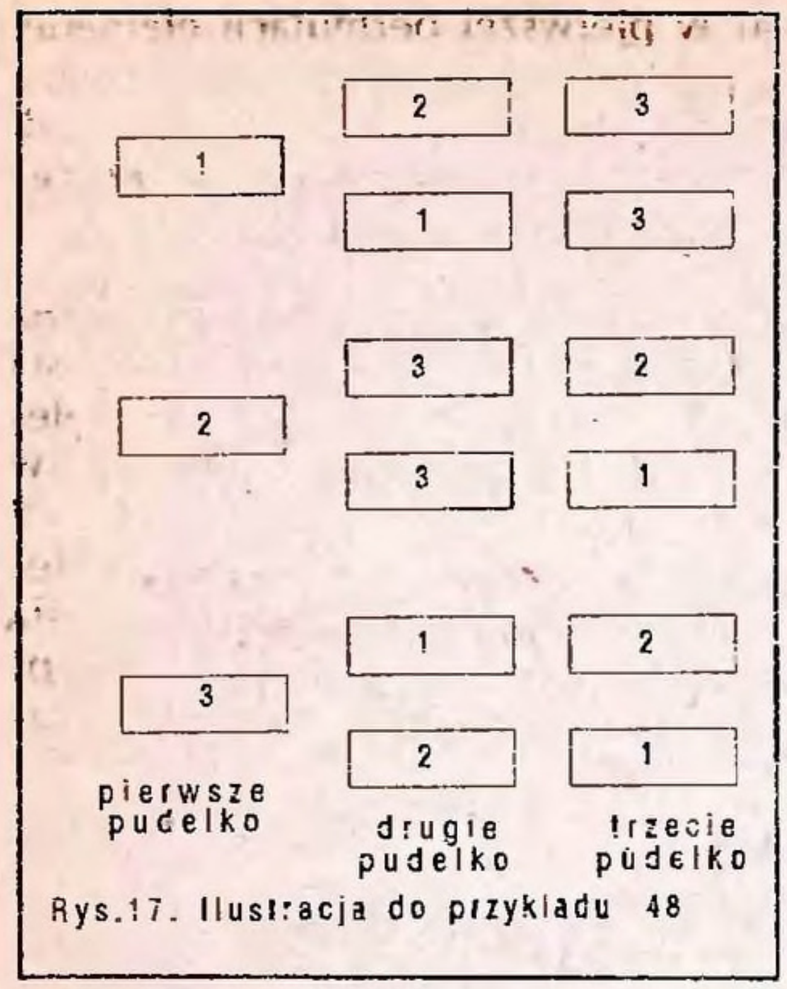
W przykładzie 47 zamieszczono program generowania wszystkich permutacji zbioru {*1,...,n*}. Permutacją zbioru n-elementowego nazywamy każdy różnowartościowy ciąg elementów zbioru **X** o długości *n*. W programie wykorzystano metodę generowania permutacji w tzw. porządku antyleksykograficznym, w którym ciąg permutacji zbioru {*1,...,*n} ma następujące własności:

a) w pierwszej permutacji elementy występują w rosnącej kolejności, a w ostatniej — w malejącej, czyli ostatnia permutacja jest odwróceniem pierwszej,

b) podany ciąg permutacji można podzielić na *n* bloków o długości *(n — 1) !*. Odpowiadają one malejącym wartościom elementu w ostatniej pozycji. Pierwsze *n — 1* pozycji bloku zawierającego element *p* w ostatniej pozycji określa ciąg permutacji zbioru {*1,...,n*}\{*p*} w porządku antyleksykograficznym.

Przykład 47

```
PROGRAM prog41;
 {$a-}
 CONST
  maxil=10;
 TYPE
  ciag=ARRAY[1..maxil] OF integer;
 VAR
  per:ciag;
  n,il,i:integer;
 PROCEDURE przestaw(a,b:integer);
  VAR
   elp:integer;
  BEGIN
   elp:=per[a];
   per[a]:=per[b];
   per[b]:=elp;
  END;
 PROCEDURE drukujper;
  VAR
   i:integer;
   zn:string[1];
  BEGIN
   IF il>20 THEN
    BEGIN
     READ(zn);
     CLRSCR;
     il:=1;
    END;
   FOR i:=1 TO n DO
    WRITE(per[i],' ');
   WRITELN;
   il:=il+1;
  END;
 PROCEDURE odwrocenie(m:integer);
  VAR
   i,j:integer;
  BEGIN
   i:=1;
   j:=m;
   WHILE i<j DO
    BEGIN
     przestaw(i,j);
     i:=i+1;
     j:=j-1;
    END;
  END;
 PROCEDURE nowaper(m:integer);
  VAR
   i:integer;
  BEGIN
   IF m=1 THEN drukujper
   ELSE
    FOR i:=1 TO m DO
     BEGIN
      nowaper(m-1);
      IF i<m THEN
       BEGIN
        przestaw(i,m);
        odwrocenie(m-1);
       END;
     END;
  END;
BEGIN
 WRITELN;
 WRITELN('PODAJ WARTOSC n<=',maxil);
 REPEAT
  READ(n);
  IF n>maxil THEN
   WRITELN('NIEPOPRAWNA WARTOSC n -',
           ' PODAJ NOWA WARTOSC n')
 UNTIL n<=maxil;
 CLRSCR;
 il:=1;
 FOR i:=1 TO n DO
  per[i]:=i;
 nowaper(n);
END.
```



Rys.17. Ilustracja do przykładu 48

W programie **prog41** wyodrębniono cztery procedury:

— przestaw *(a,b)* — zamiana miejscami elementów a i *b* w tablicy *perm* zawierającej permutację;
— odwrócenie *(m)* — zamiana kolejności *m* pierwszych elementów tablicy *perm*;
— *nowaperm (m)* — rekurencyjne utworzenie kolejnej *m*-elementowej permutacji;
— *drukujperm* — wyświetlenie bieżącej permutacji.

Po wyświetleniu kolejnych 20 permutacji program oczekuje na wciśnięcie klawisza **ENTER** w celu wyświetlenia dalszych permutacji.

W procedurach tych wykorzystano tzw. **zmienną globalną** *perm*, a w procedurze *drukujperm* także zmienną *il*, zadeklarowaną na poziomie programu (poza procedurami). Deklaracje zmiennych globalnych muszą wystąpić przed deklaracjami procedur, w których są wykorzystywane. Występujący w programie zapis {**$a** —} jest niezbędny dla programów rekurencyjnych.

W kolejnym przykładzie zamieszczono program rozwiązujący zadanie wyznaczania wszystkich możliwych rozmieszczeń *k* obiektów w *n* pudełkach, tak aby w kazdym pudełku znalazł się jeden obiekt *($k \leq n$)*. Zauważmy, że dla $k = n$ zadanie sprowadza się do wyznaczania n-elementowych permutacji. Dla $k < n$ zadanie jest bardziej złożone. Metodę wyznaczania takich rozmieszczeń prześledzimy dla $k = 2$ *i* $n = 3$. Można przyjąć następujący sposób postępowania:

a) spośród wszystkich pudełek wybieramy jedno (zwane dalej pierwszym) i wkładamy do niego pierwszy obiekt,
b) spośród dwóch pozostałych pudełek wybieramy jedno (zwane dalej drugim) i wkładamy do niego drugi obiekt,
c) nie zmieniając pierwszego pudełka, przekładamy drugi obiekt do następnego pudełka.

Wszystkie możliwe rozmieszczenia uzyskamy powtarzając czynności a — c dla każdego z możliwych sposobów wyboru pierwszego pudełka. Numerując pudełka od 1 do 3 i przyjmując zasadę, że spośród wolnych pudełek wybieramy pudełko o najmniejszym numerze, uzyskamy kolejne rozmieszczenia obiektów jak na rysunku 17.

Jeżeli przez **P** oznaczymy zbiór numerów pudełek — $P = \{1,...n\}$ — to nasze zadanie sprowadza się do znalezienia wszystkich ciągów różnowartościowych $< p_1, p_2 ... p_k >$ o wyrazach ze zbioru **P** (są to tzw. k-elementowe wariacje bez powtórzeń ze zbioru n-elementowego). Wyraz p_1 takiego ciągu możemy wybrać na *n* sposobów, wyraz p_2 na $n - 1$ sposobów. Przy ustalonych wyrazach $p_1, p_2 ... p_{i-1}$ możemy jako p_i wybrać dowolny spośród $n - i + 1$ elementów zbioru $\mathbf{P} \setminus \{p_1, p_2, ... p_{i-1}\}$

Przykład 48

```
PROGRAM prog42;
 {$a-}
 CONST
  nn=9;
  kk=7;
 TYPE
  elem=ARRAY[1..nn] OF char;
  kontr=ARRAY[1..nn] OF boolean;
  sukc=ARRAY[1..kk] OF char;
 VAR
  dowybr:elem;
  praw:kontr;
  wynik:sukc;
  i,n,k,liczw:integer;
PROCEDURE wyprowadz;
 VAR
  m:integer;
  zn:char;
 BEGIN
  IF liczw<20 THEN
   BEGIN
    READ(zn);
    clrscr;
    liczw:=2;
   END
  ELSE
   liczw:=liczw+1;
  FOR m:=1 TO k-1 DO
   WRITE(wynik[m]);
  WRITELN(wynik[k]);
 END;
PROCEDURE wybierz(i,n,k:integer; dowybr:elem;
                  praw:kontr; VAR wynik:sukc);
 VAR
  l:integer;
 BEGIN
  IF i<k THEN
   BEGIN
    FOR l:=1 TO n DO
     IF praw[l] THEN
      BEGIN
       wynik[i]:=dowybr[l];
       praw[l]:=false;
       wybierz(i+1,n,k,dowybr,praw,wynik);
       praw[l]:=true;
      END;
   END
  ELSE
   FOR l:=1 TO n DO
    IF praw[l] THEN
     BEGIN
      wynik[k]:=dowybr[l];
      wyprowadz;
     END;
 END;
BEGIN
 REPEAT
  WRITELN;
  WRITELN('podaj k i N (k<=N)');
  READLN(k,n);
 UNTIL k<=n;
 WRITE('podaj elementy zbioru : ');
 FOR i:=1 TO n DO
  BEGIN
   praw[i]:=true;
   READ(dowybr[i]);
  END;
 clrscr;
 WRITELN;
 WRITE(' ',k,'-elementowe wariacje z ',
       n,'-elementowego zbioru:');
 FOR i:=1 TO n DO
  WRITE(' ',dowybr[i]);
 WRITELN;
 WRITELN;
 liczw:=1;
 wybierz(1,n,k,dowybr,praw,wynik);
END.
```

W przykładzie wykorzystano dwa nowe typy zmiennych:

— *char* — którego wartościami są wszystkie reprezentacje znaków oraz kodów sterujących występujących na klawiaturze mikrokomputera; dla naszych celów można uznać, że jest on równoważny deklaracji *string [1]*;
— *boolean* — którego wartościami są stałe logiczne *true i false*.

ELEMENTY GRAFIKI KOMPUTEROWEJ

4



Grafika komputerowa jest bardzo ważną dziedziną informatyki rozwijaną w wielu liczących się ośrodkach. Jej celem jest wypracowanie efektywnych programów (standardów) umożliwiających łatwe i szybkie generowanie obrazów figur i brył z uwzględnieniem punktu widzenia i napotykanych przeszkód.

W ramach niniejszego opracowania nie jest możliwe przedstawienie żadnego znanego standardu graficznego, a nawet przedstawienie zasad generowania prostych obrazów. Naszym celem jest jedynie zaprezentowanie koncepcji kreślenia kolorowych obrazów w komputerze AMSTRAD CPC 6128 w oparciu o pakiet graficzny CPC 6128 — Grafik Version 1.0 firmy Heinsoeth Software Munich (wybrane problemy grafiki komputerowej prezentowane są w numerach „IKS-a", w cyklu „Grafika na ekranie").

Ekran monitora dzieli się na dwie odrębnie wysterowane, niezależne od siebie części:

— obszar główny, wykorzystywany do zapisu tekstów i określenia obrazów, zwany dalej kartką;
— pobocze.

Pobocze nie jest wykorzystywane do zapisu tekstów i kreślenia obrazów,

można natomiast zmieniać jego kolor. Kolor pobocza stanowi tło dla kartki. Do zmiany koloru pobocza wykorzystywana jest procedura.

border *(i,j)*

której parametrami są identyfikatory kolorów. Programista dysponuje 27 kolorami, oznaczanymi numerami od 0 do 26. Kolory przyporządkowane poszczególnym numerom zamieszczono w tabeli 2.

W procedurze **border** podaje się numery dwóch kolorów zmienianych kilka razy w ciągu sekundy. Jeżeli oba numery są jednakowe, to efekt migotania jest niedostrzegalny.

Program umożliwiający zmianę koloru pobocza (bez migotania) zamieszczono w przykładzie 49.

Przykład 49

```
PROGRAM prog43;
 VAR
  i:integer;
  ch:char;
  {$i grafik3.inc}
 BEGIN
  FOR i:=0 TO 26 DO
   BEGIN
    border(i,i);
    REPEAT UNTIL keypressed;
    READ(kbd,ch);
   END;
  border(1,1);
 END.
```

Program ten po każdorazowym naciśnięciu dowolnego klawisza zmienia kolor pobocza na odpowiadający kolejnemu numerowi. Do sprawdzenia czy klawisz został wciśnięty wykorzystano funkcję standardową **keypressed,** przyjmującą wartość *true* po wciśnięciu klawisza i *false* po odczytaniu kodu wciśniętego klawisza (**READ** (*kbd,...*). Po wyczerpaniu wszystkich możliwych kolorów, kolor pobocza zmienia się na niebieski, przyjmowany standardowo przez system. Efekt migotania można uzyskać realizując program z przykładu 50, w którym kolejne pary kolorów pobocza są ustalane przez użytkownika.

Przykład 50

```
PROGRAM prog44;
 VAR
  k1,k2:integer:
  ch:char;
  {$i grafik3.inc}
 BEGIN
  REPEAT
   WRITELN('zmiana koloru ramki (T/N)?');
   READ(kbd,ch);
   IF (ch='T') OR (ch='t') THEN
    BEGIN
     WRITELN('podaj nr pierwszego koloru');
     READLN(k1);
     WRITELN('podaj nr drugiego koloru');
     READLN(k2);
     border(k1,k2);
    END;
  UNTIL (ch='N') OR (ch='n');
  border(1,1);
 END.
```

Bardziej skomplikowane są zasady wykorzystywania kartki. Można na niej zapisywać teksty **piórem tekstowym** oraz kreślić obrazy **piórem graficznym.** Przed użyciem pióra należy wskazać **kałamarz,** z którego czerpany będzie tusz. W każdym kałamarzu można umieścić jednocześnie — w dwóch odrębnych jego częściach — dwa tusze (różnokolorowe lub tego samego koloru). Jeżeli tusz będzie czerpany z kałamarza, w którym umieszczono różnokolorowe tusze, uzyskamy efekt migotania jak w przypadku pobocza.

Liczba dostępnych kałamarzy uzależniona jest od trybu wykorzystania ekranu:
— 16, w trybie 0 (numerowanych od 0 do 15),
— 4, w trybie 1 (numerowanych od 0 do 3),
— 2, w trybie 2 (numerowanych od 0 do 1).

W miarę wzrostu numeru trybu wykorzystania ekranu rośnie jakość kreślonego obrazu (rozdzielczość) kosztem zmniejszającej się liczby kałamarzy możliwych do jednoczesnego wykorzystania, a tym samym liczby kolorów.

Do napełniania kałamarzy wykorzystuje się procedurę

ink *(nrk, i, j)*

gdzie:
nrk — numer kałamarza (który powinien uwzględnić tryb wykorzystania ekranu),
i,j — numery kolorów tuszu (wg tabeli zamieszczonej na obudowie każdego CPC 6128).

W przykładzie 51 przedstawiono procedurę napełniania kałamarzy wg żądań użytkownika. Procedura ta pozwala na wypełnienie tylko takiej ilości kałamarzy, jaka wynika z trybu wykorzystania ekranu.

Przykład 51

```
PROCEDURE napelnij(tryb:integer);
 VAR
  il,k1,k2,
  i:integer;
  ch:char;
 BEGIN
  IF tryb=0 THEN il:=16
  ELSE
   IF tryb=1 THEN il:=4
   ELSE il:=2;
  i:=0;
  REPEAT
   WRITELN('zmiana tuszu w ');
   WRITELN('kalamarzu nr',i);
   WRITELN('(T/N)?');
   READ(kbd,ch);
   IF (ch='T') OR (ch='t') THEN
    BEGIN
     WRITELN('nr pierwszego koloru');
     READLN(k1);
     WRITELN('nr drugiego koloru');
     READLN(k2);
     ink(i,k1,k2);
    END;
    i:=i+1;
  UNTIL i=il;
END;
```

Jak już podano, do zapisu tekstów oraz kreślenia obrazów wykorzystywane są dwa różne pióra: tekstowe i graficzne. Z każdym z nich związane są oddzielne procedury.

Ich omawianie rozpoczniemy od procedur związanych z piórem tekstowym.

Użycie pióra tekstowego powinno być poprzedzone wyborem kałamarza, którego tusze będą wykorzystywane do zapisu tekstu. Służy do tego procedura.

pen *(nrk)*

gdzie: *nrk* — numer kałamarza.

Wszystkie litery zapisywane są na tle, którego kolor wybierany jest przez wykonanie procedury

paper *(nrk)*

gdzie: *nrk* — numer kałamarza.

Jeżeli procedury te nie zostaną użyte przed wykorzystaniem pióra to dla tła przydzielany jest standardowo kałamarz 0, a dla pióra — kałamarz 1.

Dla ilustracji działania procedur **paper** i **pen** w przykładzie 52, po ustaleniu trybu wykorzystania ekranu i napełnieniu kałamarzy, następuje wybieranie kolejnych kałamarzy dla tła. W przykładzie 53 — po czynnościach wstępnych — następuje wybieranie kolejnych kałamarzy dla pióra.

Przykład 52

```
PROGRAM prog45;
 VAR
  tryb,ilosc,nrk:integer;
  ch:char;
  {$i grafik3.inc}
  {$i p51}
 BEGIN
  WRITELN('podaj tryb');
  READLN(tryb);
  napelnij(tryb);
  IF tryb=0 THEN ilosc:=16
  ELSE
   IF tryb=1 THEN ilosc:=4
   ELSE ilosc:=2;
  mode(tryb);
  FOR nrk:=0 TO ilosc-1 DO
   BEGIN
    paper(nrk);
    WRITELN;
    WRITELN('0123456789012345678?');
    REPEAT UNTIL keypressed;
    READ(kbd,ch);
   END;
  mode(2);
  ink(0,1,1);
  paper(0);
  ink(1,26,26);
  pen(1);
 END.
```

Przykład 53

```
PROGRAM prog46;
 VAR
  tryb,ilosc,nrk:integer;
  ch:char;
  {$i grafik3.inc}
  {$i p51}
 BEGIN
  WRITELN('podaj tryb');
  READLN(tryb);
  napelnij(tryb);
  IF tryb=0 THEN ilosc:=16
  ELSE
   IF tryb=1 THEN ilosc:=4
   ELSE ilosc:=2;
  mode(tryb);
  FOR nrk:=0 TO ilosc-1 DO
   BEGIN
    pen(nrk);
    WRITELN;
    WRITELN('0123456789012345678?');
    REPEAT UNTIL keypressed;
    READ(kbd,ch);
   END;
```

```
  mode(2);
  ink(0,1,1);
  paper(0);
  ink(1,26,26);
  pen(1);
END.
```

W obu podanych przykładach do ustalenia żądanego trybu wykorzystania ekranu zastosowano procedurę:

mode *(nrtryb)*

gdzie

nrtryb — przyjmuje wartości 0,1,2

Należy wyjaśnić, że z trybem wykorzystania ekranu wiąże się również liczba znaków tekstu, które mogą być wyświetlane w jednym wierszu. Wynosi ona:

— 20, dla trybu 0,
— 40, dla trybu 1,
— 80, dla trybu 2.

Przed zakończeniem każdego z programów przywracany jest standardowy tryb wykorzystania ekranu (2), standardowe numery kałamarzy tła (0) i pióra (1) oraz standardowe numery tuszów w tych kałamarzach.

Program z przykładu 54 umożliwia prześledzenie wszystkich przedstawionych właściwości pióra tekstowego.

Przykład 54

```
PROGRAM prog47;
 VAR
  tryb,nrk:integer;
  ch:char;
  {$i grafik3.inc}
  {$i p51}
 BEGIN
  WRITELN('podaj tryb');
  READLN(tryb);
  mode(tryb);
  WRITELN;
  REPEAT
   WRITELN('zmiana tuszu w ');
   WRITELN('kalamarzach (T/N)?');
   READ(kbd,ch);
   IF (ch='T') OR (ch='t') THEN
     napelnij(tryb);
   WRITELN('zmiana tla (T/N)?');
   READ(kbd,ch);
   IF (ch='T') OR (ch='t') THEN
    BEGIN
     WRITELN('podaj nr kalamarza');
     READLN(nrk);
     paper(nrk);
    END;
   WRITELN('zmiana piora (T/N)?');
   READ(kbd,ch);
   IF (ch='T') OR (ch='t') THEN
    BEGIN
     WRITELN('podaj nr kalamarza');
     READLN(nrk);
     pen(nrk);
    END;
  UNTIL ch=' ';
  mode(2);
  ink(0,1,1);
  paper(0);
  ink(1,26,26);
  pen(1);
END.
```

Z piórem graficznym związanych jest znacznie więcej procedur.

Przed użyciem pióra graficznego niezbędne jest wywołanie bezparametrowej procedury

initgrafik

ustalającej początkowe, standardowe warunki pracy. Po zakończeniu wykorzystywania pióra graficznego należy wywołać bezparametrową procedurę

leavegrafik

Odpowiednikiem procedury *pen* dla pióra graficznego jest procedura

grafpen *(nrk)*

a odpowiednikiem procedury **paper** — procedura

grafpaper *(nrk)*

Zasady wykorzystywania kałamarzy przez pióra graficzne są identyczne, jak dla pióra tekstowego.

Kartka podzielona jest na 400x640 punktów (niezależnie od trybu wykorzystania ekranu), tj. 400 wierszy (numerowanych od 0 do 399) po 640 punktów (numerowanych od 0 do 639) w każdym z nich. Punkt można zakreślić za pomocą procedury.

plot *(x,y)*

gdzie:

x — nr wiersza,
y — nr punktu w wierszu

Od ostatnio ustalonego położenia pióra można wykreślić odcinek do punktu wskazanego w procedurze

draw *(x,y)*

gdzie:

x,y — współrzędne punktu, do którego będzie kreślony odcinek.

Dodajmy, że punkt wskazany w draw staje się ostatnio ustalonym położeniem pióra graficznego. W przykładzie 55 zamieszczono program kreślenia prostokąta wyznaczającego ramy kartki.

Przykład 55

```
PROGRAM prog48;
 {$i grafik3.inc}
 BEGIN
  ink(0,0,0);
  ink(1,19,19);
  initgrafik;
  grafpaper(0);
  grafpen(1);
  plot(0,0);
  draw(0,399);
  draw(639,399);
  draw(639,0);
  draw(0,0);
  REPEAT UNTIL keypressed;
  leavegrafik;
  ink(0,1,1);
  ink(1,26,26);
 END.
```

W kolejnym przykładzie zamieszczono prosty program kreślenia prostokąta o współrzędnych wierzchołków (100,100), (100,250), (550,250) (550,100) wraz z przekątnymi, kreślonymi innymi kolorami.

Przykład 56

```
PROGRAM prog49;
 {$i grafik3.inc}
 BEGIN
  initgrafik;
  mode(0);
  ink(2,0,0);
  ink(3,12,26);
  ink(4,16,16);
  ink(5,11,11);
  grafpaper(2);
  clg(2);
  grafpen(5);
  plot(100,100);
  draw(100,250);
  draw(550,250);
  draw(550,100);
  draw(100,100);
  grafpen(3);
  draw(550,250);
  grafpen(4);
  plot(100,250);
  draw(550,100);
  REPEAT UNTIL keypressed;
  leavegrafik;
 END.
```

W programie **prog49** wykorzystano procedurę

clg *(nrk)*

„czyszczącą" kartkę w kolorze ustalonym w ostatnio wykonanej procedurze **grafpaper** i ustawiającą nowy numer kałamarza dla tła pióra graficznego.

Następne przykłady ilustrują sposób kreślenia figur graficznych oraz niektóre problemy związane z projektowaniem bardziej złożonych obrazów (ograniczając się do płaszczyzny).

W przykładzie 57 przedstawiono jeden ze sposobów kreślenia okręgu. Wykorzystano w nim metodę polegającą na kolorowaniu punktów położonych na okręgu w odległości kątowej co jeden stopień.

Przykład 57

```
PROGRAM prog50;
{$i grafik3.inc}
 PROCEDURE okrag(xs,ys,r:integer);
  VAR
   x,y,i:integer;
   wsp:real;
  BEGIN
   wsp:=pi/180;
   FOR i:=0 TO 359 DO
    BEGIN
     x:=round(r*sin(i*wsp));
     y:=round(r*cos(i*wsp));
     plot(xs+x,ys+y);
    END;
  END;
 BEGIN
  initgrafik;
  ink(0,0,0);
  grafpaper(0);
  clg(0);
  grafpen(1);
  okrag(320,200,100);
  REPEAT UNTIL keypressed;
  leavegrafik;
  ink(0,1,1);
 END.
```

Wrażenie obracania się koła uzyskano w programie z przykładu 58 poprzez zmianę kolorów szprych koła. Efekt ten jest namiastką bardzo dynamicznie rozwijającej się problematyki animacji obrazów.

Przykład 58

```
PROGRAM prog51;
{$i grafik3.inc}
 VAR
  i,j:integer;
  ch:char;
  wsp:real;
 PROCEDURE okrag(xs,ys,r:integer);
  VAR
   x,y,i:integer;
   wsp:real;
  BEGIN
   wsp:=pi/180;
   FOR i:=0 TO 359 DO
    BEGIN
     x:=round(r*sin(i*wsp));
     y:=round(r*cos(i*wsp));
     plot(xs+x,ys+y);
    END;
  END;
BEGIN
 initgrafik;
 mode(1);
 ink(2,0,0);
 grafpaper(2);
 clg(2);
 ink(0,12,12);
 ink(3,10,10);
 grafpen(0);
 okrag(200,200,150);
 wsp:=pi/180;
 j:=0;
 FOR i:=0 TO 29 DO
  BEGIN
   grafpen(3);
   plot(200,200);
   draw(200+round(149*sin(6*j*wsp)),
        200+round(149*cos(6*j*wsp)));
   j:=j+1;
   grafpen(1);
   plot(200,200);
   draw(200+round(149*sin(6*j*wsp)),
        200+round(149*cos(6*j*wsp)));
   j:=j+1;
  END;
 WHILE NOT keypressed DO
  BEGIN
   ink(1,10,10);
   ink(3,26,26);
   FOR i:=1 TO 4000 DO;
   ink(3,10,10);
   ink(1,26,26);
   FOR i:=1 TO 4000 DO;
  END;
 leavegrafik;
 ink(1,26,26);
 ink(0,1,1);
END.
```

Inny problem to kolorowanie figur. W programie **prog52** z przykładu 59 zilustrowano jeden ze sposobów kolorowania pola prostokąta.

Przykład 59

```
PROGRAM prog52;
 {$i grafik3.inc}
  VAR
   i,nr:integer;
   ch:char;
 BEGIN
  initgrafik;
  mode(0);
  ink(0,0,0);
  grafpaper(0);
  clg(0);
  grafpen(1);
  plot(100,100);
  draw(100,250);
  draw(550,250);
  draw(550,100);
  draw(100,100);
  REPEAT
   gotoxy(0,23);
   WRITE('podaj nr kalamarza  ');
   gotoxy(19,23);
   READ(nr);
   grafpen(nr MOD 16);
   FOR i:=100 TO 250 DO
    BEGIN
     plot(100,i);
     draw(550,i);
    END;
   REPEAT UNTIL keypressed;
   READ(kbd,ch);
  UNTIL ch=' ';
  ink(0,1,1);
  mode(2);
  leavegrafik;
END.
```

W programie wykorzystano instrukcję

gotoxy *(x,y)*

umożliwiającą ustawienie położenia pióra tekstowego na określonym miejscu ekranu. Ekran widziany jest tu jako tablica znaków o wymiarach 25 wierszy i liczbie kolumn zależnej od trybu wykorzystania ekranu.

Wszędzie tam, gdzie zamieszcza się obrazy obiektów w przestrzeni trójwymiarowej pojawia się problem likwidowania rysunków niewidocznych (przesłoniętych) części obiektów, ścian i krawędzi.

Najprostszą ilustracją tego problemu jest kreślenie nakładających się częściowo prostokątów, przedstawione w przykładzie 60.

Przykład 60

```
PROGRAM prog53;
 {$i grafik3.inc}
 VAR
  x11,y11,x12,y12,
  x21,y21,x22,y22:integer;
 BEGIN
  WRITELN('podaj wspolrzedne dolnego ',
          'prostokata');
  READLN(x11,y11,x12,y12);
  WRITELN('podaj wspolrzedne gornego ',
          'prostokata');
  READLN(x21,y21,x22,y22);
  initgrafik;
  mode(0);
  ink(0,0,0);
  clg(0);
  grafwindow(x11,y11,x12,y12);
  grafpaper(4);
  clg(4);
  grafwindow(x21,y21,x22,y22);
  grafpaper(8);
  clg(8);
  REPEAT UNTIL keypressed;
  mode(2);
  ink(0,1,1);
  leavegrafik;
END.
```

W programie wykorzystano właściwości tzw. okien graficznych wyznaczanych za pomocą procedury

grafwindow *(x_l,y_l,x_p,y_p)*

gdzie:

x_l,y_l — współrzędne lewego dolnego wierzchołka okna,

x_p,y_p, — współrzędne prawego górnego wierzchołka okna.

Okno graficzne jest prostokątem. Przyjmuje się, że dla pióra graficznego kartka jest początkowym, standardowym oknem graficznym.

W podanym przykładzie każdy prostokąt reprezentowany jest w postaci okna, a do ich kolorowania wykorzystano znaną już instrukcję **clg.** Dotyczy ona ostatnio określonego okna. Efekt przesłaniania uzyskano sterując kolejnością kolorowania prostokątów. W praktycznie spotykanych zadaniach uzyskanie tego efektu wymaga skomplikowanych obliczeń i należy do najbardziej złożonych problemów grafiki komputerowej.

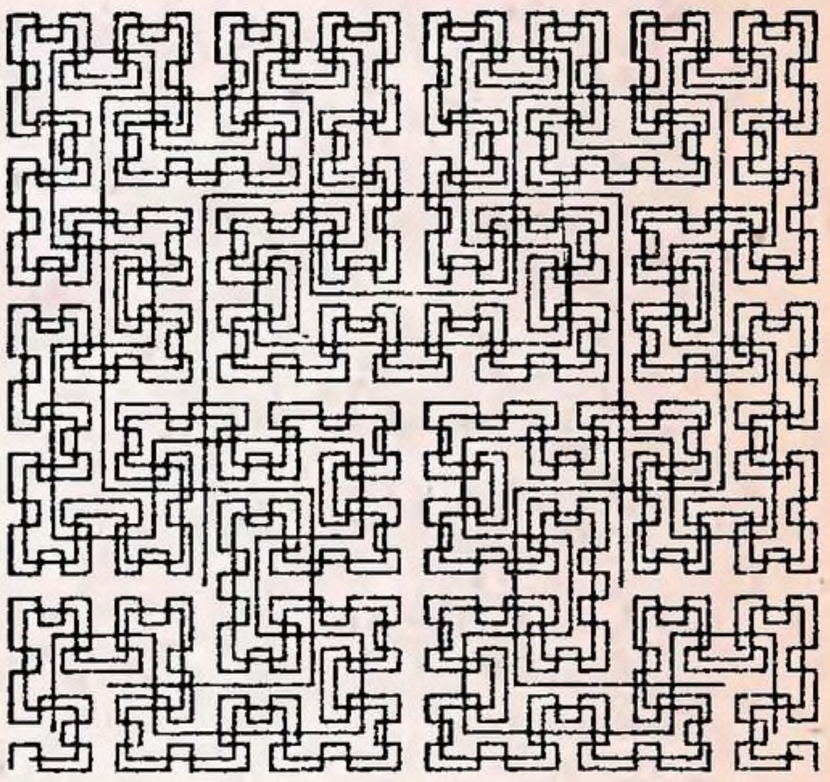
W przykładzie 61 okna graficzne wykorzystano do wyświetlenia kolorów tuszów, którymi standardowo napełnione są poszczególne kałamarze.

Przykład 61

```
PROGRAM prog54;
 {$i grafik3.inc}
VAR
 x1,x2,y1,y2,
 i,j,k:integer;
BEGIN
 x1:=10;
 x2:=60;
 y1:=50;
 y2:=150;
 initgrafik;
 mode(0);
 ink(0,0,0);
 grafpaper(0);
 clg(0);
 k:=1;
 FOR i:=0 TO 1 DO
  FOR j:=0 TO 7 DO
   BEGIN
    grafwindow(x1+j*80,y1+i*125,
               x2+j*80,y2+i*125);
    clg(k);
    k:=k+1;
   END;
 REPEAT UNTIL keypressed;
 ink(0,1,1);
 ink(1,26,26);
 mode(2);
 leavegrafik;
END.
```

Podane przykłady na pewno nie oddają w pełni złożoności problematyki grafiki komputerowej. Ilustrują one jedynie wybrane, najczęściej spotykane problemy.

Na zakończenie zamieszczamy przykład ilustrujący wykorzystanie rekurencji do kreślenia krzywych, zwanych krzywymi Hilberta. (poniżej)



Atrakcyjny układ graficzny pokazany na poniższym rysunku powstał ze złożenia pięciu krzywych. (poniżej)



Procedura **A** jest wywoływana w programie głównym tylko raz dla każdej z nakładanych na siebie krzywych Hilberta. Program główny określa również punkt początkowy krzywej,

tj. początkowe wartości zmiennych *x* i *y* oraz przyrost jednostkowy *h*. Przez *h* oznaczono tu całą szerokość strony, na której będą kreślone krzywe.

Ponieważ w języku PASCAL obowiązuje zasada, że pierwsze wystąpienie obiektu powinno być jego definicją, zastosowano tzw. deklaracje wyprzedzające procedur B,C i D. W deklaracji takiej podaje się pełny nagłówek procedury, a blok procedury zastępuje się słowem kluczowym **FORWARD.** W dalszej części programu deklaruje się blok procedury poprzedzony jej identyfikatorem (parametrów nie podaje się).

Przykład 62

```
PROGRAM prog55;
   {program kresli krzywe Hilberta }
   {zadanego rzedu                 }
{$a-}
{$i grafik3.inc}
 CONST
   h0=256;
 VAR
   nrk,i,h,x,y,x0,y0,n:integer;
 PROCEDURE b(i:integer); FORWARD;
 PROCEDURE c(i:integer); FORWARD;
 PROCEDURE d(i:integer); FORWARD;
 PROCEDURE ustawpioro;
  BEGIN
   plot(x,y);
  END;
 PROCEDURE kresl;
  BEGIN
   draw(x,y);
  END;
 PROCEDURE a(i:integer);
  BEGIN
   IF i>0 THEN
    BEGIN
     d(i-1); x:=x-h; kresl;
     a(i-1); y:=y-h; kresl;
     a(i-1); x:=x+h; kresl;
     b(i-1)
    END
  END;
 PROCEDURE b;
  BEGIN
   IF i>0 THEN
    BEGIN
     c(i-1); y:=y+h; kresl;
     b(i-1); x:=x+h; kresl;
     b(i-1); y:=y-h; kresl;
     a(i-1)
    END
  END;
 PROCEDURE c;
  BEGIN
   IF i>0 THEN
    BEGIN
     b(i-1); x:=x+h; kresl;
     c(i-1); y:=y+h; kresl;
     c(i-1); x:=x-h; kresl;
     d(i-1)
    END
  END;
 PROCEDURE d;
  BEGIN
   IF i>0 THEN
    BEGIN
     a(i-1); y:=y-h; kresl;
     d(i-1); x:=x-h; kresl;
     d(i-1); y:=y+h; kresl;
     c(i-1)
    END
  END;
BEGIN
 initgrafik;
 i:=0; h:=h0;
 x0:=(h DIV 2)+75;
 y0:=x0+50;
 READ(n);
 mode(0);
 ink(3,0,0);
 grafpaper(3);
 clg(3);
 ink(4,11,11);
 ink(5,7,7);
 ink(6,12,12);
 ink(7,19,19);
 ink(8,16,16);
 nrk:=3;
 REPEAT
  nrk:=nrk+1;
  i:=i+1; h:=h DIV 2;
  x0:=x0+(h DIV 2);
  y0:=y0+(h DIV 2);
  x:=x0; y:=y0;
  grafpen(nrk);
  ustawpioro;
  a(i);
 UNTIL i=n;
 REPEAT UNTIL keypressed;
 mode(2);
 leavegrafik
END.
```

Zakończenie

W niniejszym opracowaniu zostały omówione jedynie niektóre właściwości języka PASCAL. Autorzy sądzą, że pozwoli to tworzyć nieskomplikowane programy osobom, które opanowały matematykę na poziomie szkoły średniej, a w wielu przypadkach — tylko podstawowej.

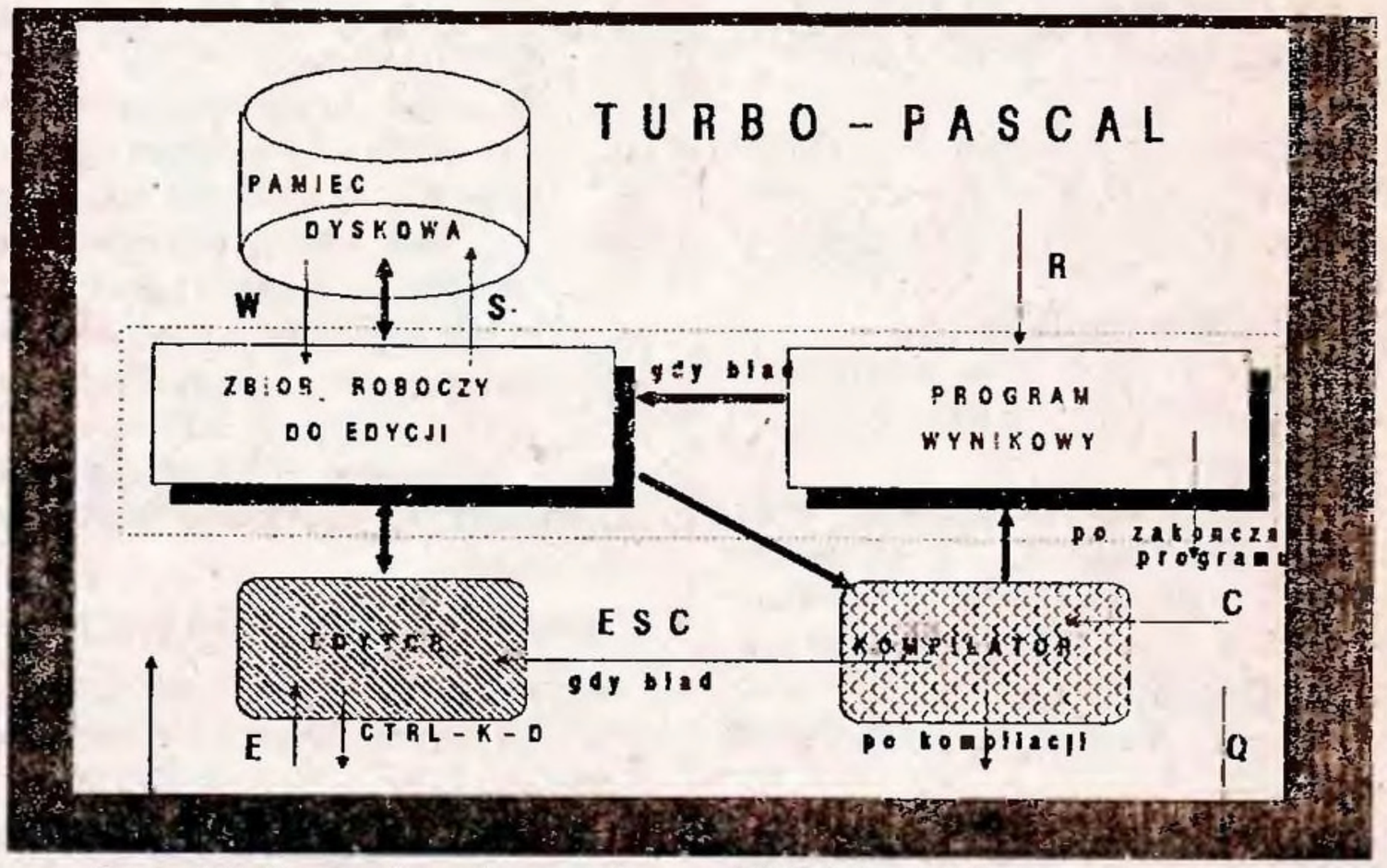
Adam KAPICA
Stefan ROZMUS

Dodatek

TURBO PASCAL jest systemem programowania obejmującym:

— edytor programów źródłowych, umożliwiających przygotowanie tekstu programu,
— kompilator, tłumaczący tekst programu na postać zrozumiałą przez komputer,
— programy pomocnicze, realizujące między innymi uruchamianie programu, składowanie tekstu programu w pamięci dyskowej oraz ładowanie tekstu programu z pamięci dyskowej do systemu.

Strukturę TURBO PASCAL — zilustrowano na rys. 18



Rys.18.Organizacja systemu TURBO-PASCAL:
→ - przepływy danych ⟶ - przepływy sterowania(komendy)

Uwidoczniono tu podstawowe komendy systemu (akcje wywoływane przez użytkownika) przypisując je odpowiednim elementom. Wejście do systemu TURBO PASCAL następuje w wyniku wywołania komendy TURBO. Z poziomu systemu CP/M PLUS (lub CP/M-80 2.2). Powrót do systemu CP/M jest efektem komendy Q wywołanej z poziomu TURBO PASCAL. Na poziomie systemu TURBO PASCAL możliwe jest:

— wejście do edytora (komenda **E**);
— wywołanie kompilatora (komenda **C**, która powinna być podana po uprzednim przygotowaniu tekstu programu);
— uruchomienie programu, po zakończeniu kompilacji (komenda **R**);
— przesłanie tekstu programu do pamięci dyskowej (komenda **S**), lub
— załadowanie tekstu programu z pamięci dyskowej do zbioru.

Wyjście z edytora następuje po jednoczesnym wciśnięciu klawiszy <**CTRL**> i **K** a następnie **D.** Wyjście z kompilatora następuje po zakończeniu kompilacji programu lub po wykryciu błędu. Jeżeli kompilator wykrył błąd w programie źródłowym (sygnalizując go użytkownikowi), po wciśnięciu klawisza <**ESC**> następuje automatyczne wejście do edytora wraz ze wskazaniem miejsca występowania błędu w tekście programu. Wyjście z wykonania programu następuje po jego zakończeniu lub po wykryciu błędu (wykonania programu).

Jak uruchomić pierwszy program?

W tym celu należy wprowadzić z klawiatury komendę TURBO i wcisnąć

klawisz RETURN. Spowoduje to wyświetlenie komunikatu wstępnego:

— *komunikat wstępny* —

zawierającego pytanie o sposób sygnalizowania błędów kompilacji. Na pytanie to należy dać odpowiedź wciskając klawisz **Y** lub **N.** Wciśnięcie klawisza **N** spowoduje, że błędy kompilacji będą sygnalizowane wyłącznie numerami. Wciśnięcie klawisza **Y** sprawi, że sygnalizacja błędów będzie zawierała również słowne ich opisy. Po wprowadzaniu odpowiedzi (zalecamy odpowiedź **Y**) zostaje wyświetlone tzw. menu główne, zawierające zbiór dostępnych komend systemu TURBO PASCAL:

— menu główne —

Po wejściu do systemu TURBO PASCAL należy wywołać edytor (E). Używając komend edytora wprowadzić program źródłowy do zbioru roboczego, skompilować go (C) i uruchomić (R) lub — jeśli wystąpi błąd — przejść ponownie do edycji (wciśnięcie klawisza <**ESC**>. Ciąg czynności prowadzących do uruchomienia programu można przedstawić w postaci następującego algorytmu:
a) wcisnąć klawisz **W** (żądanie wybrania zbioru roboczego),
b) podać nazwę zbioru roboczego edytora i wcisnąć klawisz **RETURN,**
c) wcisnąć klawisz **E** (wywołanie edytora),
d) wprowadzić z klawiatury wiersz tekstu programu źródłowego (zostanie on wyświetlony na ekranie monitora),
e) wcisnąć klawisz **RETURN,**
f) czynności d) i e) wykonać dla kolejnych wierszy tekstu programu źródłowego,
g) po wprowadzeniu ostatniego wiersza tekstu programu źródłowego wcisnąć jednocześnie klawisze <**CTRL**> i **K,** a następnie wcisnąć klawisz **D** (wyjście z edytora),
h) wcisnąć klawisz **C** (wykonanie kompilacji),
i) jeśli nie wystąpi błąd kompilacji wcisnąć klawisz **R** (uruchomienie programu), a w przypadku przeciwnym wcisnąć klawisz **ESC** (ponowne wywołanie edytora w celu poprawienia programu. Po jego poprawieniu przejść do punktu **h**).

Po wciśnięciu klawisza **E** nastąpi wyświetlenie komunikatu:

Work file name:

po którym należy podać nazwę zbioru (zawierającego tekst programu) w postaci:

nazwa lub *nazwa. typ*

gdzie:
nazwa — ciąg od jednego do ośmiu znaków określających nazwę zbioru,
typ — ciąg od jednego do trzech znaków określających typ zbioru (pominięcie typu jest równoważne przyjęciu typu domyślnego PAS).
W obu przypadkach żaden ze znaków nie powinien być znakiem:

= ? * . , < > : ; [ ] { }

Po podaniu nazwy zbioru (np. FILENAME.PAS) i wciśnięciu klawisza RETURN w pierwszym wierszu, u góry ekranu, zostanie wyświetlony opis stanu edytora w postaci

Line 1 Col i Insert Indent A:FILENAME.PAS

Od tego momentu edytor reaguje tylko na własne komendy, a nie na komendy systemu TURBO PASCAL. W następnych wierszach będą wyświetlane kolejne wersy wprowadzanego tekstu programu źródłowego. Gdy cały ekran zostanie zapełniony, wprowadzenie nowego wiersza następuje po przesunięciu zawartości ekranu (bez opisu stanu edytora) o jeden wiersz w górę. Nie należy się jednak obawiać utraty „znikających" wierszy. Można je bowiem wyświetlić wykorzystując odpowiednie komendy edytora.

Wciśnięcie klawisza C (po wyjściu z edytora, po wprowadzeniu tekstu programu) spowoduje wykonanie kompilacji programu.

Jeżeli zakończy się ona poprawnie, program można uruchomić przez wciśnięcie klawisza R. W razie błędnej kompilacji należy ponownie wywołać edytor (tym razem przez wciśnięcie klawisza <**ESC**> i wykorzystując niżej opisane komendy edytora poprawić program źródłowy. Czynności poprawianie—kompilacja wykonujemy aż do uzyskania bezbłędnej kompilacji.

WYBRANE KOMENDY EDYTORA

Pierwsze próby uruchomienia programu z reguły kończą się niepowodzeniem. Przyczyny są następujące:
— w programie źródłowym wystąpił zapis niezgodny z regułami języka PASCAL (sygnalizowany jest błąd kompilacji),
— podczas wykonania programu wystąpił błąd, np. próba wczytania danej o postaci niedopuszczalnej dla typu zmiennej, której ma być przypisana (sygnalizowany jest błąd wykonania),
— w programie wystąpiły błędy logiczne (program został zrealizowany lecz wyniki są niezgodne z oczekiwanymi).

W każdym z tych przypadków należy wywołać edytor przez wciśnięcie klawisza:
— <**ESC**> — po sygnalizacji błędu kompilacji lub wykonania: na ekranie monitora zostanie wyświetlony program źródłowy, a kursor wskaże miejsce, w którym wystąpił błąd,
— E — gdy stwierdziliśmy błędy logiczne: na ekranie monitora zostanie wyświetlony program źródłowy, a kursor będzie wskazywał pierwszy znak w pierwszej linii programu.

Wykorzystując komendy edytora program można poprawić i uruchomić ponownie. Edytor dostarcza 45 komend, z których opiszemy kilka, naszym zdaniem niezbędnych.

Zapisujemy je symbolicznie następująco

<**CTRL**> — **ZN**

Oznacza on, że po przy wciśniętym klawiszu **CONTROL** należy nacisnąć klawisz ze znakiem „**ZN**".

Poprawienie programu może polegać m.in. na:
— wstawieniu nowych znaków w wierszu,
— skasowaniu określonych znaków w wierszu,
— wstawieniu nowego wiersza do programu,
— skasowaniu określonego wiersza programu.

Edytor ma dwa tryby wstawiania znaków w wierszu:
INSERT oraz OVERWRITE. W trybie INSERT nowy znak zostaje wprowadzony na pozycję kursora, a pozostała część wiersza (od kursora w prawo do końca wiersza) jest przesuwana w prawo. W trybie OVERWRITE nowy znak zostaje wprowadzony na miejsce znaku wskazywanego przez kursor. Zmianę trybu wstawiania znaków powoduje komenda <**CTRL**> — **V.** Do skasowania znaku znajdującego się przed kursorem wykorzystywany jest klawisz <DEL>. Do przesunięcia kursora można wykorzystać jedną z komend:
<**CTRL**>—**D** — przesunięcie kursora w prawo o jeden znak,
<**CTRL**>—**S** — przesunięcie kursora w lewo o jeden znak,
<**CTRL**>—**E** — przesunięcie kursora o jeden wiersz w górę,
<**CTRL**>—**X** — przesunięcie kursora o jeden wiersz w dół,
<**CTRL**>—**W** — tzw. przewinięcie w górę (przesunięcie tekstu wyświetlonego na ekranie o jeden wiersz w dół),
<**CTRL**>—**Z** — tzw. przewinięcie w dół (przesunięcie tekstu wyświetlonego na ekranie o jeden wiersz w górę).

Wstawienie nowego wiesza umożliwia komenda <**CTRL**>—**N**. Powoduje ona wstawienie nowego wiersza (za wierszem, w którym znajduje się

kursor). Znaki, od pozycji kursora w prawo do końca wiersza, zostają przeniesione do nowego wiersza. Pozycja kursora się nie zmienia. Do kasowania wiersza, w którym znajduje się kursor, służy komenda **<CTRL>—Y.** Po skasowaniu wiersza tekst znajdujący się poniżej kasowanego wiersza jest przesuwany o jeden wiersz w górę. Kursor przesuwa się na pozycję pierwszego znaku. Przy stosowaniu tej komendy należy zachować ostrożnośc, gdyż nie ma możliwości odzyskania skasowanego wiersza.

PRZECHOWYWANIE ZBIORÓW ROBOCZYCH

Żądanie utworzenia nowego zbioru roboczego, gdy w pamięci istnieje już jakiś zbiór (np. poddawany edycji), powoduje wyprowadzenie pytania dotyczącego składowania istniejącego zbioru. Odpowiedź twierdząca (Y) powoduje w tym przypadku przechowanie zbioru w pamięci dyskowej, przecząca zaś — jego zniszczenie poprzez otworzenie nowego zbioru roboczego. Widzimy zatem, że składowanie zbioru zapobiega jego zniszczeniu. Składowanie programu można uzyskać również przez wykonanie komendy S. Jeżeli istnieje poprzednia wersja zbioru o tej samej nazwie, co składowany zbiór roboczy, to typ zbioru w tej wersji zostanie przemianowany na BAK. Wprowadzając nowy program o dużej ilości wierszy źródłowych warto dokonywać jego składowania po wprowadzeniu kolejnej porcji wierszy. Zapobiegnie to utracie całego wprowadzonego programu w przypadku jakiejkolwiek awarii. Wczytanie przechowanego programu umożliwia znana już komenda W.

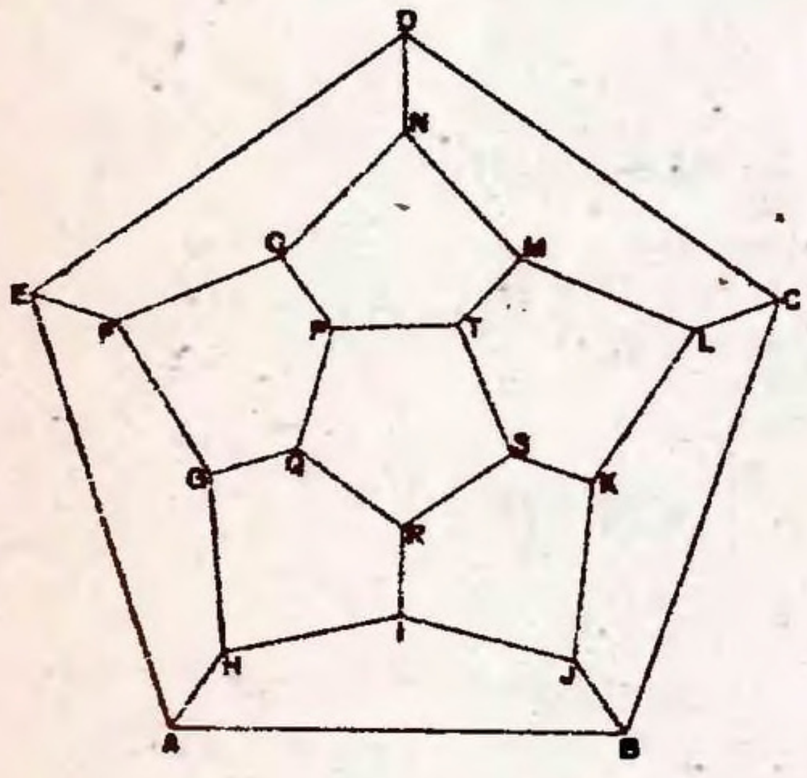
W „Dodatku" omówiono jedynie te właściwości systemu TURBO PASCAL, które są niezbędne do uruchomienia przykładowych programów. Bardziej zaawansowani programiści powinni zapoznać się ze wszystkimi właściwościami systemu, omówionymi w dokumentacji producenta.

Prezentujemy dwa bardziej złożone programy, w których wykorzystano własności pakietu graficznego GRAFIKA 3.INC.

GRA HAMILTONA

William Hamilton, matematyk i astronom angielski z pierwszej połowy XIX wieku, obmyślił ciekawą rozrywkę zwaną „podróżą po dodekaedrze" (dwunastościanie). W „podróży" tej chodzi o jednorazowe odwiedziny wszystkich naroży dwunastościanu z powróceniem do naroża, z którego się wyszło.

Program umożliwia „podróż" lub demonstruje jej przebieg, przy czym dwunastościan został zastąpiony następującą figurą płaską:



Klawisze funkcyjne:

0 — wyjście z programu,
1 — powrót do „menu",
2 — demonstracja gry,
3 — gra,
A÷T — wybór odpowiedniego punktu figury,
TAB — cofnięcie ruchu,

UWAGA: program kończy się, gdy następuje powrót do punktu początkowego; wygrywa ten, kto przejdzie przez wszystkie punkty zgodnie z podanymi zasadami.

```
PROGRAM grahamiltona;
 {$i grafik3.inc}
 TYPE
  nazwypunktow=(a,b,c,d,e,f,g,h,i,j,k,
                l,m,n,o,p,q,r,s,t);
  opispunktu=RECORD
               x,y:integer;
               wskp:boolean;
               sasiedzi:ARRAY[1..3] OF
                 RECORD
                  poprz,
                  lewy,
                  prawy:nazwypunktow
                 END
             END;
  dodekaedr=ARRAY[nazwypunktow] OF
                              opispunktu;
  lzn=string[40];
VAR
  fig:dodekaedr;
  ppocz:nazwypunktow;
  ch:char;
  pozlit:ARRAY['A'..'T'] OF
          RECORD
           x,
           y:integer
          END;
  wykgr:boolean;


{*****************************************}
{*   procedury pomocnicze do manipulacji *}
{*   zobrazowaniem graficznym            *}
{*                                       *}
 PROCEDURE punkt(nazp:nazwypunktow);
  BEGIN
   plot(fig[nazp].x,fig[nazp].y)
  END;
 PROCEDURE linia(nazp:nazwypunktow);
  BEGIN
   draw(fig[nazp].x,fig[nazp].y)
  END;
 PROCEDURE litera(lit:char);
  BEGIN
   tag;
   plot(pozlit[lit].x,pozlit[lit].y);
```

```
    grafout(lit);
    tagoff
  END;
  PROCEDURE napis(l:lzn;nrlin,
                  nrkol:integer);
    VAR
     i,j,k:integer;
    BEGIN
     i:=nrkol*16-16;
     j:=400-nrlin*16;
     tag;
     FOR k:=1 TO length(l) DO
      BEGIN
       plot(i,j);
       grafout(l[k]);
       i:=i+16
      END;
     tagoff
    END;
{*****************************************}
{*****************************************}
{*      ustawienie wspolrzednych         *}
{*                                       *}
 PROCEDURE ustawwsp;
  PROCEDURE wspp(nazp:nazwypunktow;
                 xp,yp:integer);
   BEGIN
    WITH fig[nazp] DO
     BEGIN x:=xp; y:=yp; wskp:=false END
   END;
  BEGIN
   wspp(a,200,50);  wspp(b,400,50);
   wspp(c,464,240); wspp(d,300,356);
   wspp(e,136,240); wspp(f,176,226);
   wspp(g,216,162); wspp(h,222,84);
   wspp(i,300,100); wspp(j,378,84);
   wspp(k,384,162); wspp(l,424,226);
   wspp(m,354,258); wspp(n,300,314);
   wspp(o,246,258); wspp(p,270,228);
   wspp(q,254,174); wspp(r,300,140);
   wspp(s,346,174); wspp(t,330,228)
  END;
{*****************************************}
{*****************************************}
{*    opisy wierzcholkow sasiednich      *}
{*                                       *}
 PROCEDURE ustawsas;
  PROCEDURE sd(nazp,p1,p2,
               p3:nazwypunktow);
   BEGIN
    WITH fig[nazp] DO
     BEGIN
      WITH sasiedzi[1] DO
       BEGIN
        poprz:=p1; lewy:=p2; prawy:=p3
       END;
      WITH sasiedzi[2] DO
       BEGIN
        poprz:=p2; lewy:=p3; prawy:=p1
       END;
      WITH sasiedzi[3] DO
       BEGIN
        poprz:=p3; lewy:=p1; prawy:=p2
       END
     END
   END;
  BEGIN
   sd(a,e,h,b); sd(b,a,j,c); sd(c,b,l,d);
   sd(d,c,n,e); sd(e,d,f,a); sd(f,e,o,g);
   sd(g,f,q,h); sd(h,g,i,a); sd(i,h,r,j);
   sd(j,i,k,b); sd(k,j,s,l); sd(l,k,m,c);
   sd(m,l,t,n); sd(n,m,o,d); sd(o,n,p,f);
   sd(p,o,t,q); sd(q,p,r,g); sd(r,q,s,i);
   sd(s,r,t,k); sd(t,s,p,m)
  END;
{*****************************************}
```

```
{*****************************************}
{*     wypelnienie tablicy opisujacej    *}
{*     polozenie nazw wierzcholkow       *}
{*                                       *}
PROCEDURE nazwyw;
 PROCEDURE wp(l:char;x,y:integer);
  BEGIN
   pozlit[l].x:=x;
   pozlit[l].y:=y
  END;
 BEGIN
  wp('A',195,40);  wp('B',395,40);
  wp('C',475,245); wp('D',295,376);
  wp('E',115,245); wp('F',174,246);
  wp('G',195,160); wp('H',225,78);
  wp('I',295,90);  wp('J',363,78);
  wp('K',392,160); wp('L',420,246);
  wp('M',361,276); wp('N',278,329);
  wp('O',231,276); wp('P',250,230);
  wp('T',317,247); wp('Q',244,164);
  wp('S',350,188); wp('R',307,139)
 END;
{*****************************************}


{*****************************************}
{*        wyswietlenie figury            *}
{*                                       *}
PROCEDURE wyswfig;
 VAR
  i,j:nazwypunktow;
  l:char;
 BEGIN
  border(0,0); grafpaper(3);
  clg(3); grafpen(0);
  punkt(a);
  FOR i:=a TO d DO
   linia(succ(i));
  linia(a);
  punkt(f);
  FOR i:=g TO o DO
   linia(i);
  linia(f);
  linia(e);
  j:=h;
  FOR i:=a TO d DO
   BEGIN
    punkt(i);
    linia(j);
    j:=succ(succ(j))
   END;
  punkt(p);
  FOR i:=q TO t DO
   linia(i);
  linia(p);
  linia(o);
  j:=g;
  FOR i:=q TO t DO
   BEGIN
    punkt(i);
    linia(j);
    j:=succ(succ(j))
   END;
  nazwyw;
  grafpen(2);
  FOR l:='A' TO 'T' DO
   litera(l)
 END;

{*****************************************}


{*****************************************}
{*  pokaz wyznaczania drogi hamiltona    *}
{*                                       *}
```

```
PROCEDURE pokaz;
 VAR
  akt,pop,pom:nazwypunktow;
  i1,i,i1,i2:integer;
 FUNCTION nre:integer;
  VAR zn:boolean;
      i:integer;
  BEGIN
   zn:=false;
   i:=1;
   REPEAT
    WITH fig[akt] DO
      IF sasiedzi[i].poprz=pop THEN
       BEGIN nre:=i; zn:=true END;
    i:=i+1
   UNTIL zn
  END;
 PROCEDURE wlewo;
  BEGIN
   pom:=akt;
   akt:=fig[akt].sasiedzi[nre].lewy;
   pop:=pom;
   linia(akt)
  END;
PROCEDURE wprawo;
 BEGIN
  pom:=akt;
  akt:=fig[akt].sasiedzi[nre].prawy;
  pop:=pom;
  linia(akt)
 END;
PROCEDURE opoznij;
 VAR i:integer;
 BEGIN
  i:=1;
  REPEAT
   i:=i+1
  UNTIL i>20000
 END;
BEGIN
 ustawwsp;
 ustawsas;
 wyswfig;
 opozni j;
 i1:=random(19);
 akt:=a;
 IF i1>0 THEN
  FOR i:=0 TO i1-1 DO
   akt:=succ(akt);
pop:=fig[akt].sasiedzi[random(2)+1]
     .poprz;
grafpen(1);
litera(chr(ord('A')+ord(akt)));
opozni j;
punkt(akt);
FOR i1:=1 TO 2 DO
 BEGIN
  FOR i2:=1 TO 3 DO
   BEGIN wlewo; opozni j END;
  FOR i2:=1 TO 3 DO
   BEGIN wprawo; opozni j END;
  wlewo; opozni j;
  wprawo; opozni j;
  wlewo; opozni j;
  wprawo; opozni j;
 END;
 napis('KONIEC POKAZU',24,1);
 wykgr:=false
END;
{*****************************************}

{*****************************************}
{*                menu                   *}
{*                                       *}
 PROCEDURE menu;
  BEGIN
   border(0,0); grafpaper(3);
   clg(3); grafpen(2);
   napis('GRA HAMILTONA',2,5);
   napis('MENU:',4,5);
   napis('0 - WYJSCIE Z PROGRAMU',6,10);
   napis('1 - POWROT DO MENU',8,10);
   napis('2 - POKAZ',10,10);
   napis('3 - GRA',12,10);
   wykgr:=false
  END;
{*****************************************}

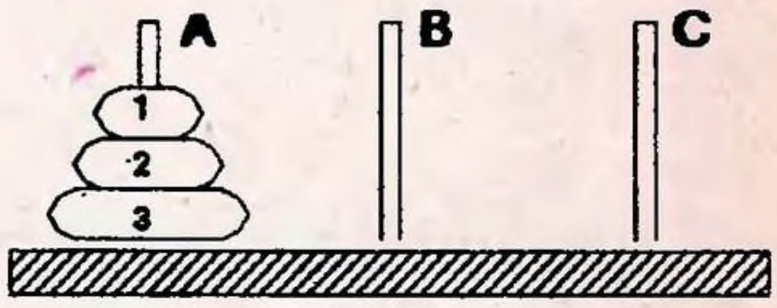
{*****************************************}
{*                gra                    *}
{*                                       *}
 PROCEDURE gra;
  VAR
   koniec:boolean;
   ruchy:ARRAY[1..20] OF nazwypunktow;
   ppocz,pnast:nazwypunktow;
   nrakt:integer;
  FUNCTION nazp(ch:char):nazwypunktow;
   VAR
    i,j:integer;
    p:nazwypunktow;
   BEGIN
    IF (ch>='A') AND (ch<='T') THEN
      i:=ord(ch)-ord('A')
    ELSE
      i:=ord(ch)-ord('a');
    p:=a;
    FOR j:=1 TO i DO
     p:=succ(p);
    nazp:=p
   END;
 PROCEDURE wygaskom;
  VAR
   lpom:lzn;
   i:integer;
  BEGIN
   grafpen(3);
   FOR i:=1 TO 40 DO
    lpom[i]:=' ';
   napis(lpom,24,1);
   grafpen(1)
  END;
 FUNCTION rpopr(pnast:nazwypunktow):
          boolean;
  BEGIN
   WITH fig[pnast] DO
    IF (NOT weko) AND
     ((sasiedzi[1].poprz=ruchy[nrakt]) OR
      (sasiedzi[2].poprz=ruchy[nrakt]) OR
      (sasiedzi[3].poprz=ruchy[nrakt]))
     THEN rpopr:=true
     ELSE rpopr:=false
  END;
BEGIN
 ustawwsp;
 ustawsas;
 wyswfig;
 wykgr:=true;
 koniec:=false;
 grafpen(1);
 napis('PODAJ PUNKT POCZATKOWY',24,1);
 REPEAT
  REPEAT UNTIL keypressed;
  READ(kbd,ch);
 UNTIL ((ch>='A') AND (ch<='T')) OR
       ((ch>='a') AND (ch<='t'));
 ppocz:=nazp(ch);
 IF (ch>='a') AND (ch<='t') THEN
                ch:=chr(ord(ch)-32);
 litera(ch);
 nrakt:=1;
 ruchy[1]:=ppocz;
 wygaskom;
 napis('PODAJ NASTEPNY PUNKT',24,1);
 punkt(ppocz);
```

```
WHILE NOT ((ch=#48) OR (ch=#49) OR
  (ch=#50) OR (ch=#51) OR koniec) DO
 BEGIN
  REPEAT UNTIL keypressed;
  READ(kbd,ch);
  IF ch=#9 THEN
   BEGIN
    IF nrakt>1 THEN
    BEGIN
     grafpen(0);
     linia(ruchy[nrakt-1]);
     fig[ruchy[nrakt]].wskp:=false;
     nrakt:=nrakt-1;
     wygaskom;
     napis('PODAJ NASTEPNY PUNKT',24,1);
     punkt(ruchy[nrakt]);
    END
   END
  ELSE
   IF ((ch>='A') AND (ch<='T')) OR
      ((ch>='a') AND (ch<='t')) THEN

    BEGIN
     pnast:=nazp(ch);
     IF rpopr(pnast) THEN
      BEGIN
       linia(pnast);
       IF pnast=ppocz THEN koniec:=true
       ELSE
        BEGIN
         nrakt:=nrakt+1;
         ruchy[nrakt]:=pnast;
         fig[pnast].wskp:=true;
         wygaskom;
         napis('PODAJ NASTEPNY PUNKT',
                 24,1);
         punkt(ruchy[nrakt]);
        END
      END
     ELSE
      BEGIN
       wygaskom;
       napis(
        'ZLY RUCH-PODAJ NASTEPNY PUNKT',
        24,1);
       punkt(ruchy[nrakt])
      END
    END;
  END;
   IF koniec THEN
    BEGIN
     wygaskom;
     IF nrakt=20 THEN
      napis(
       'KONIEC GRY. WYGRALES-GRATULACJE',
       24,1)
     ELSE
      napis(
       'KONIEC GRY. NIESTETY PRZEGRALES',
       24,1);
     wykgr:=false
    END
  END;
{*****************************************}
BEGIN
 crtinit;
 initgrafik;
 mode(1);
 ink(0,26,26);
 ink(1,15,15);
 ink(2,9,9);
 ink(3,0,0);
 menu;
 REPEAT
  IF NOT wykgr THEN
   BEGIN
    REPEAT UNTIL keypressed;
    READ(kbd,ch)
   END;
  CASE ch OF
   #49: menu;
   #50: pokaz;
   #51: gra
  END;
 UNTIL ch=#48;
 mode(2);
 ink(0,1,1);
 ink(1,26,26);
 border(1,1);
 leavegrafik;
END.
```

WIEŻA HANOI

Mamy trzy pręty i n krążków o różnych średnicach. Krążki mogą być nakładane na pręty, tworząc w ten sposób wieżę. Załóżmy, że *n* krążków zostało umieszczonych początkowo na pręcie A w kolejności malejących średnic (na rysunku zilustrowano tę sytuację dla $n = 3$).



Naszym zadaniem jest przesunięcie *n* krążków z pretu A na pręt C z zachowaniem pierwotnego porządku. Zasady przesuwania są następujące:

1) w każdym kroku przesuwa się dokładnie jeden krążek z jednego pręta na inny,
2) krążek nie może być nigdy nałożony na krążek o mniejszej średnicy,
3) pręt B może być użyty jako „magazyn pomocniczy".

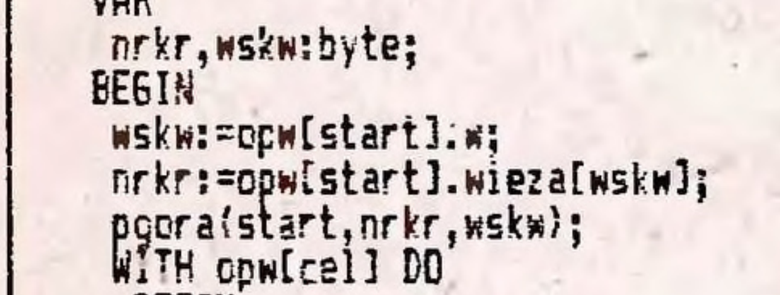
Po wprowadzeniu do programu ilości krążków (1—5) program realizuje ich przenoszenie według podanych zasad.

```
PROGRAM wiezahanoi;
 {$a-}
 {$i grafik3.inc}
  CONST
    maxilkr=5; {maksymalna ilosc krazkow}
    py=40;
    podstawa=50;
    sr1=119;
    sr2=299;
    sr3=479;
  TYPE
    znakp='a'..'c'; {nazwa preta}
    wyswiez=1..maxilkr; {maksymalna wysokosc wiezy
                         mierzona iloscia krazkow}
    opiswiez=ARRAY [znakp] OF  {opis wiez}
        RECORD
          w:wyswiez; {wskaznik wierzcholka wiezy}
          wieza:ARRAY [wyswiez] OF wyswiez
        END;
  VAR
    ilkr:integer; {zadana ilosc krazkow}
    start,  {pret, z ktorego przesuwamy krazki}
```

```
    cel,    {pret, na ktory przesuwamy krazki: }
    posr:znakp; {pret - magazyn pomocniczy    }
    opw:opiswiez;
ch1,ch2:char;nr:integer;
  PROCEDURE ustawstpocz;
     {procedura ustawiajaca poczatkowy stan wiez}
    VAR
      s:znakp: {zmienne robocze}
      i:wyswiez;
    PROCEDURE pret(x:integer);
      {procedura rysowania preta, na ktory beda}
      {nakladane krazki                        }
     BEGIN
      plot(x-3,podstawa);
      draw(x-3,270);
      plot(x+3,podstawa);
      draw(x+3,270);
     END;
  PROCEDURE krazek(i:byte);
    {procedura rysowania krazka o numerze i}
   VAR
    lin,xl,yl,xp,yp,px:integer;
  BEGIN
   grafpen(i+1);
   yl:=podstawa+(maxilkr-i)*py;
   yp:=yl+py-1;
   px:=(i-1)*10+20;
   xl:=sr1-px;
   xp:=sr1+px;
   FOR lin:=yl TO yp DO
    BEGIN
     plot(xl,lin);
     draw(xp,lin);
    END;
  END;

  BEGIN
   ink(0,0,0);
   ink(1,26,26);
   ink(2,15,15);
   ink(3,17,17);
   ink(4,6,6);
   ink(5,1,1);
   ink(6,21,21);
   grafpaper(0);
   clg(0);
   grafpen(1);
   plot(50,podstawa-1);
   draw(550,podstawa-1);
   plot(50,43);
   draw(550,43);
   plot(sr1-10,20);
   grafout('A');
   pret(sr1);
   plot(sr2-10,20);
   grafout('B');
   pret(sr2);
   plot(sr3-10,20);
   grafout('C');
   pret(sr3);
   WITH opw['a'] DO
    BEGIN
     w:=maxilkr-ilkr+1;
     FOR i:=w TO maxilkr DO
      wieza[i]:=1;
    END;
   opw['b'].w:=0;
   opw['c'].w:=0;
  FOR i:=maxilkr DOWNTO opw['a'].w DO
   krazek(i);
  FOR i:=1 TO 15000 DO
  END;
    PROCEDURE wymkr(nazw:znakp;nrkr:byte;
                    VAR sr,px:integer);
      {procedura ustalajaca wymiary i polozenie}
      {krazka na wiezy                         }
     BEGIN
      CASE nazw OF
       'a':sr:=sr1;
       'b':sr:=sr2;
       'c':sr:=sr3;
      END;
      px:=(nrkr-1)*10+20;
     END;
  FUNCTION przes(start,cel:znakp):integer;
    {funkcja wyznaczajaca wielkosc przesuniecia}
    {krazka w lewo lub w prawo                 }
   BEGIN
    CASE start OF
     'a':IF cel='b' THEN
           przes:=180
         ELSE
           przes:=360;
     'b':przes:=180;
     'c':IF cel='b' THEN
           przes:=180
         ELSE
           przes:=360;
    END;
   END;
  PROCEDURE pgora(start:znakp;nrkr,wskw:byte);
    {procedura przesuwajaca wybrany krazek}
    {z zadanej wiezy w gore               }
   VAR
    sr,px,pocz,kon,
    podst,i:integer;
   BEGIN
    wymkr(start,nrkr,sr,px);
    pocz:=sr-px;
    kon:=sr+px;
    podst:=podstawa+(maxilkr-wskw)*py;
    FOR i:=podst TO 301 DO
     BEGIN
      grafpen(0);
      plot(pocz,i);
      draw(kon,i);
      IF i<=270 THEN
       BEGIN
        grafpen(1);
        plot(sr-3,i);
        draw(sr+3,i);
       END;
      grafpen(nrkr+1);
      plot(pocz,i+py);
      draw(kon,i+py);
     END;
   END;
  PROCEDURE pdol(cel:znakp;nrkr,wskw:byte);
   {procedura przesuwajaca wybrany krazek}
   {w dol na zadana wieze                }
  VAR
   sr,px,pocz,kon,
   podst,i:integer;
  BEGIN
   wymkr(cel,nrkr,sr,px);
   pocz:=sr-px;
   kon:=sr+px;
   podst:=podstawa+(maxilkr-wskw)*py;
   FOR i:=341 DOWNTO podst+py DO
    BEGIN
     grafpen(0);
     plot(pocz,i);
     draw(kon,i);
     IF i<=270 THEN
      BEGIN
       grafpen(1);
       plot(sr-3,i);
       draw(sr+3,i);
      END;
     grafpen(nrkr+1);
     plot(pocz,i-py);
     draw(kon,i-py);
    END;
   END;
  PROCEDURE pprawo(start,cel:znakp;nrkr:byte);
```

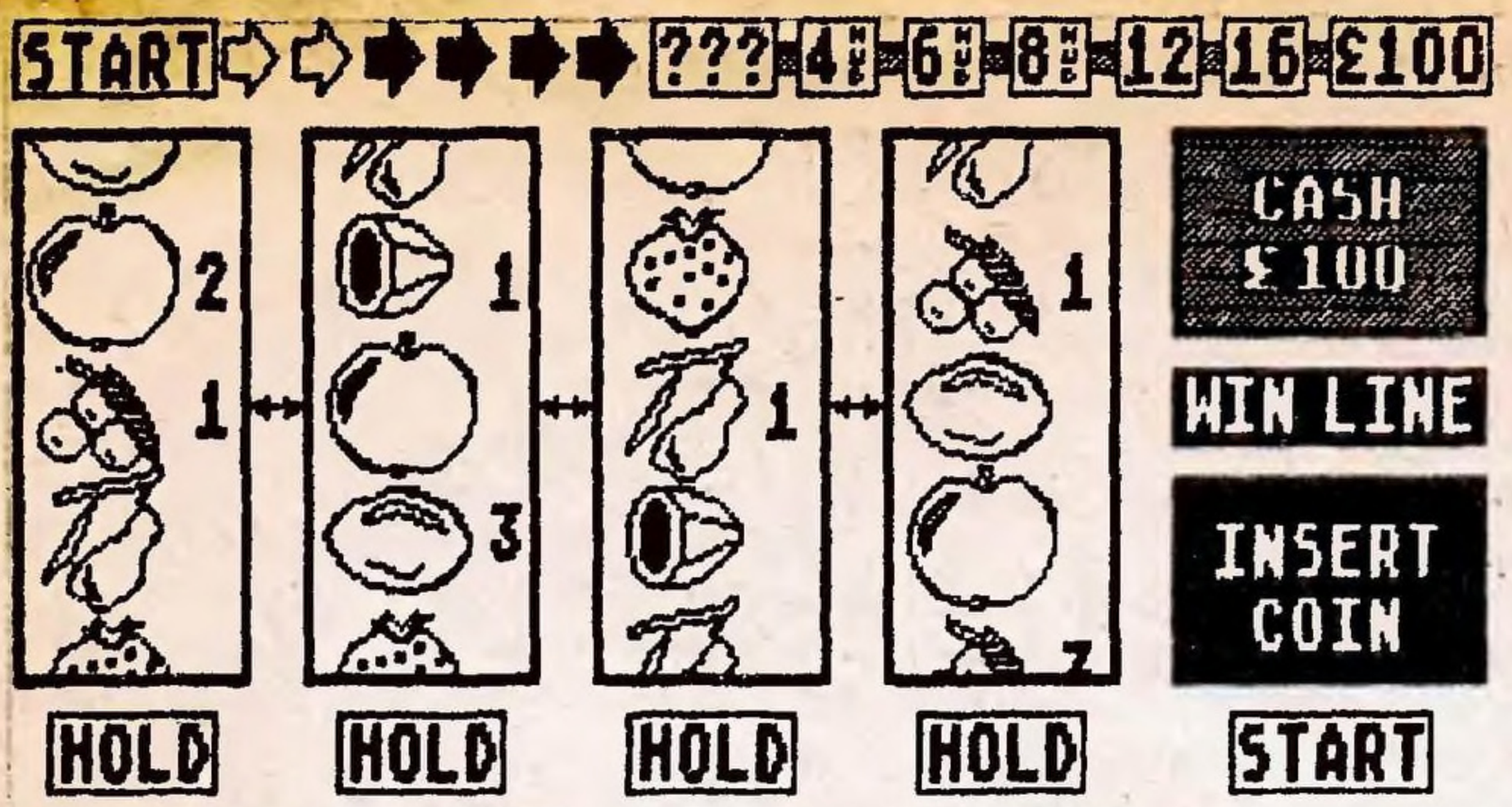
```
 {procedura przesuwajaca wybrany krazek  }
 {w prawo od wiezy 'start' do wiezy 'cel'}
VAR
 sr,px,pocz,kon,deltax,i:integer;
BEGIN
 wymkr(start,nrkr,sr,px);
 deltax:=przes(start,cel);
 pocz:=sr-px;
 kon:=sr+px-1;
 FOR i:=0 TO deltax-1 DO
  BEGIN
   grafpen(0);
   plot(pocz+i,302);
   draw(pocz+i,341);
   grafpen(nrkr+1);
   plot(kon+i,302);
   draw(kon+i,341);
  END;
END;
PROCEDURE plewo(start,cel:znakp;nrkr:byte);
 {procedura przesuwajaca wybrany krazek  }
 {w prawo od wiezy 'start' do wiezy 'cel'}
VAR
 sr,px,pocz,kon,deltax,i:integer;
BEGIN
 wymkr(start,nrkr,sr,px);
 deltax:=przes(start,cel);
 pocz:=sr-px;
 kon:=sr+px-1;
 FOR i:=0 TO deltax-1 DO
  BEGIN
   grafpen(0);
   plot(kon-i+1,302);
   draw(kon-i+1,341);
   grafpen(nrkr+1);
   plot(pocz-i+1,302);
   draw(pocz-i+1,341);
  END;
END;
PROCEDURE przen(start,cel:znakp);
   {procedura przenoszaca pojedynczy krazek}
   { z preta "start" na pret "cel"         }
VAR
 nrkr,wskw:byte;
BEGIN
 wskw:=opw[start].w;
 nrkr:=opw[start].wieza[wskw];
 pgora(start,nrkr,wskw);
 WITH opw[cel] DO
  BEGIN
   IF w=0 THEN  w:=maxilkr
   ELSE  w:=w-1;
   wieza[w]:=nrkr;
  END;
 WITH opw[start] DO
  IF w=maxilkr THEN  w:=0
  ELSE  w:=w+1;
 CASE cel OF
  'a':plewo(start,cel,nrkr);
  'b':IF start='a' THEN
        pprawo(start,cel,nrkr)
      ELSE
       plewo(start,cel,nrkr);
  'c':pprawo(start,cel,nrkr);
 END;
 wskw:=opw[cel].w;
 nrkr:=opw[cel].wieza[wskw];
 pdol(cel,nrkr,wskw);
END;
PROCEDURE przel(n:wyswiez; VAR start,posr,cel:znakp);
   {procedura przesuwajaca "n" krazkow z preta
    "start" na pret "cel" z wykorzystaniem preta
    "posr" jako magazynu}
  BEGIN
   IF n=1 THEN przen(start,cel)
   ELSE
    BEGIN
     przel(n-1,start,cel,posr);
     przen(start,cel);
     przel(n-1,posr,start,cel)
    END
  END;
BEGIN
 REPEAT
  WRITELN('podaj ilosc krazkow <= ',maxilkr);
  READLN(ilkr);
 UNTIL (ilkr>0) AND (ilkr<=maxilkr);
 initgrafik;
 mode(0);
 ustawstpocz;
 grafwindow(0,0,639,399);
 start:='a';
 cel:='c';
 posr:='b';
 przel(ilkr,start,posr,cel);
REPEAT UNTIL keypressed;
ink(1,26,26);
grafpen(1);
ink(0,1,1);
grafpaper(0);
clg(0);
mode(2);
leavegrafik;
END.
```



	2	3	4
WIŚNIE	3	5	10
CYTRYNA	3	5	10
ŚLIWKA	3	7	20
GRUSZKA	3	7	20
JABŁKO	3	10	30
TRUSKAWKA	3	10	50
ORZECH	3	15	70
CHIP	3	15	100

Hazard jest jednym z tych nałogów , który może bezpośrednio nie odbija się na zdrowiu, lecz pośrednio od - działywuje bardzo skutecznie. Iluż to ludzi popełniło samobójstwo z powodu pozostawienia swojego całe - go majątku w kasynie gry. Tobie , oczywiście, to nie grozi z prostej przyczyny, posiadasz przecież kom - puter . Tym razem daje ci on moż - ność podejmowania ryzykownych de - cyzji bez negatywnych następstw w razie pomyłki. Wbrew utartym sche - matom trzeba w tej grze również myśleć . Jako cel postawiono przed grającym zadanie zdobycia żetonów o wartości 250 punktów . Komputer przydziela ci na początku gry 100 punktów . Każdy obrót bębnów kosz - tuje ciebie żeton o wartości 1 punkta . Po " włożeniu " żetona klawiszem **SELECT** , naciskamy **START**. Wszystkie bębny wystartują , a po chwili kolejno od lewej strony zaczną się zatrzymywać . Po ich zatrzymaniu się będziesz miał do czynienia z następującymi sytuac - jami :

1. Na bębnach na wysokości strzałek są różne owoce . Punktów jest poni - żej 7 , a komputer reaguje jedynie lampką **INSERT COIN**, polecającą wrzu - cić żeton . Zrób, to naciskając **SE - LECT** . Czynność tę możesz powtó - rzyć kilkakrotnie -zwiększysz w ten sposób swój kredyt w komputerze .



Klawiszem **START** uruchomisz ponownie bębny .

2. Na bębnach są różne owoce . Punktów jest mniej niż 7 . Pod bębnami migają lampki z napisem **HOLD** . Oznacza to , że jeżeli zechcesz , to przed uruchomieniem bębnów możesz niektóre z nich zatrzymać . Dokonujesz tego klawiszami od 1 do 4 . Gdybyś w trakcie wybierania doszedł do wniosku , że zatrzymujesz niewłaściwe bębny , naciśnij **ESC** . Następnie skoryguj swój wybór . Pozostałe czynności jak w p. 1 .

3. Na wysokości strzałek na bębnach patrząc od strony lewej masz dwa , trzy lub cztery takie same owoce . Na dole w zależności od rodzaju i ilości owoców świeci się lampka z ilością punktów . Możesz je zabrać, naciskając **START**. Gdy zechcesz zdobyte punkty postawić i ewentualnie zdobyć ich więcej, naciśnij **OPTION** . Błyskające lampki po bokach , wskazujące wygrana , pokazują , ile możesz zdobyć punktów , jeśli zaryzykujesz (ta po prawej) lub , ile dostaniesz ich w postaci nagrody pocieszenia , gdy ci się nie uda (po lewej). Gdy przegrasz , komputer automatycznie dodaje punkty do twojej kasy , a następnie oczekuje na kontynuację gry . Ilość punktów , jaką możesz zdobyć w zależności od ilości i rodzaju owoców , przedstawiono w tabeli poniżej. Jak z niej widać okrągłe (CHIP) -w przeciwieństwie do rzeczywistości -jest wart najwięcej .

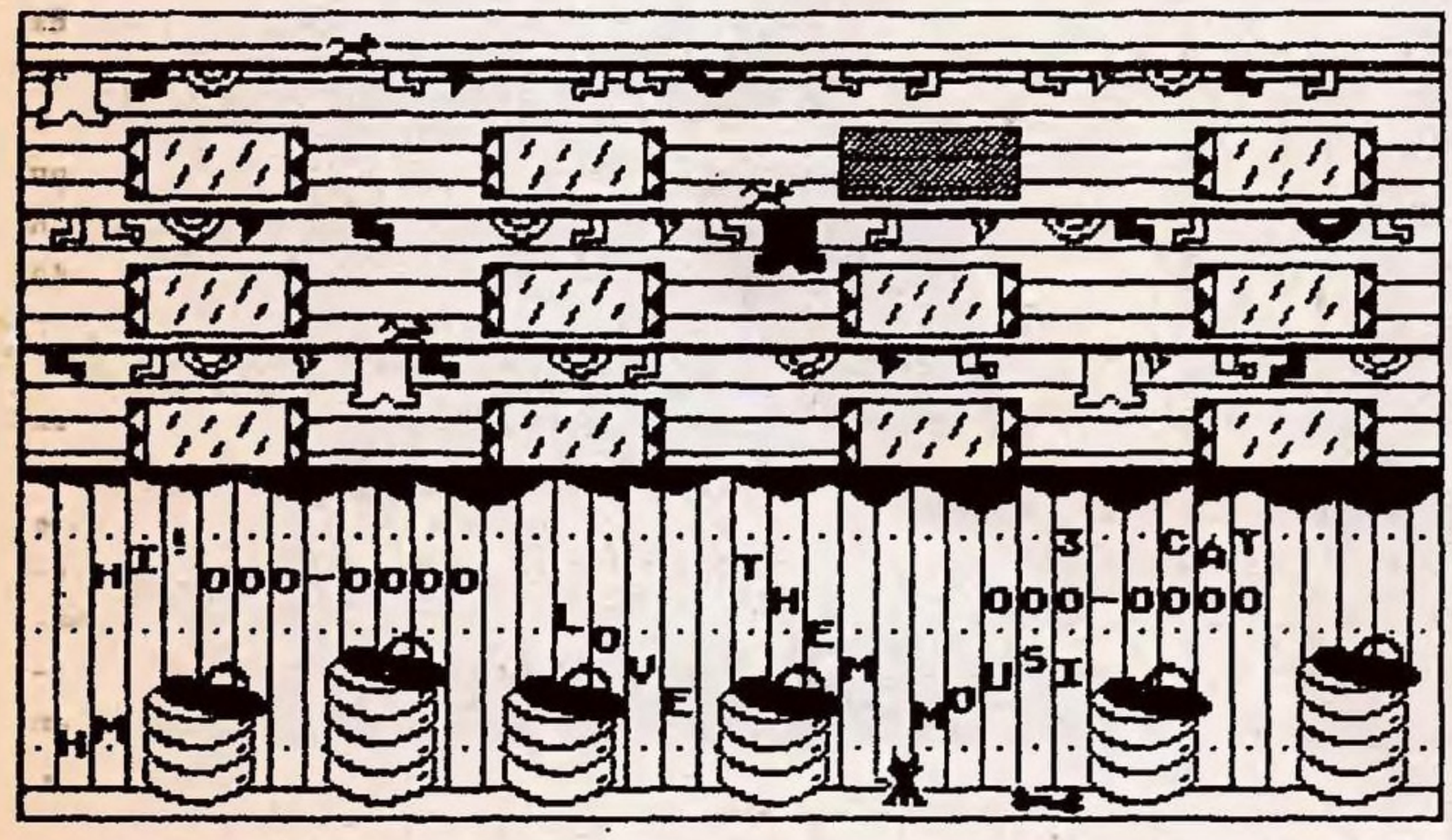
4. Gdy uzyskano na bębnach 7 punktów , komputer automatycznie wykonuje ponowny obrót bębnów . Otrzymasz ślepą wygraną , która ustawi bębny na owoce dające żetony o wartości 3 lub 5 punktów . Suma punktów zliczanych z bębnów jest pokazywana w postaci świecących się strzałek w górnej części ekranu . Punkty te nie są wartością wygranej , choć w istotny sposób wpływają na jej wielkość .

5. W przypadku , gdy na bębnach będzie więcej niż 7 punktów, otrzymujesz premię w postaci nudzy . Ilość ich jest zależna od sumy punktów na bębnach , a odczytuje się je z niebiesko świecących wskaźników, znajdujących się w górnej części ekranu . Aby zebrać te nudze w postaci punktów naciśnij **START** , a komputer uruchomi bębny , ustawiając je w możliwie atrakcyjnym położeniu . Jeżeli decydujesz się postawić do dalszej gry zdobyte nudze, przyciśnij **OPTION**. W przypadku gdybyś przegrał , zostaniesz uhonorowany nagrodą pocieszenia .Gdy wygrasz, możesz stawiać dalej aż do uzyskania maksimum , tj. 100 punktów .

Gdy uda ci się zdobyć 250 punktów lub wszystko przegrasz , będziesz mógł grać od nowa po wciśnięciu klawisza **SPACE BAR** lub **HELP**.

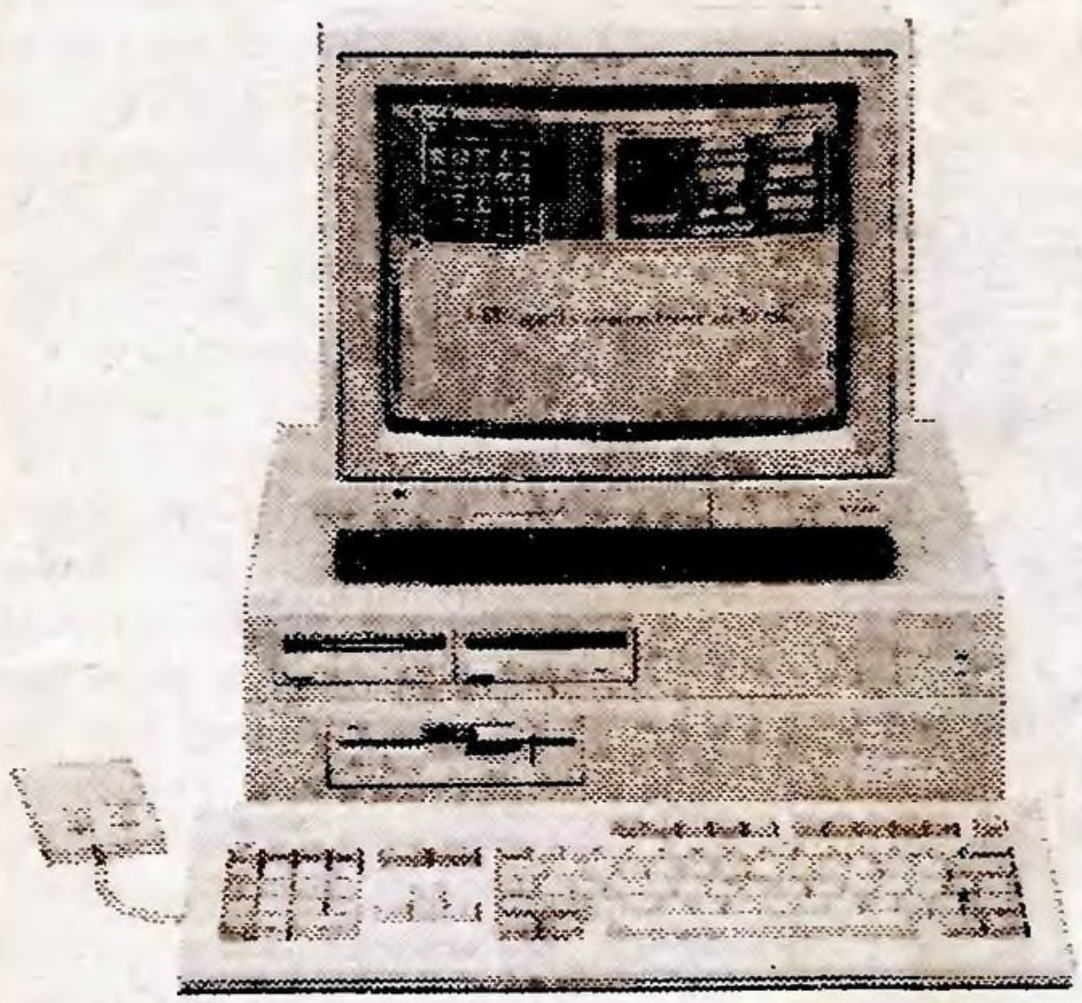


Kot jest jednym z tych zwierząt, o których słyszy się od najwłodszych lat . W przedszkolu z pewnością uczono cię wiersza " Wlazł kotek ..." , a w szkole kazano przeczytać bajkę " Kot w butach " . Nie przejmuj się , tym razem nikt nic nie będzie ci kazał . Sam wcielisz się w kota Feliksa , który będzie musiał wykazać się zwinnością i sprytem podczas pokonywania kolejnych przeszkód . Przed rozpoczęciem gry zdecyduj się , który z czterech stopni trudności wybierasz . Proponujemy pierwszy - zatytułowany KITTEN. Wyboru należy dokonywać klawiszami OPTION i SELECT . Grę uruchamia się klawiszem START lub przez naciśnięcie przycisku joystick'a . Pamiętaj o jednym : że masz tylko do wykorzystania trzy koty . Te wstrętne myszy będą robiły wszystko aby wyeliminować ciebie z gry , ale nie tylko one. Podczas próby dostania się na płot ze śmietników będzie cię starał zepchnąć biały kot. Jeżeli to mu się uda a w tym czasie przed płotem przebiegnie pies , przepadłeś . Nad płotem są rozwieszone sznurki z bielizną . Możesz po nich skakać i łapać biegające myszy . Za złapaną mysz na najwyższym sznurku otrzymujesz 90 , na środkowym 70 , a na najniższym 50 punktów . Myszy jest dużo, w związku z tym i dużo roboty . Zwróć uwagę na otwierające się co pewien czas okna ; warto w nie wskakiwać . Można zainkasować poważną ilość



punktów , oczywiście nie za darmo . Bo na przykład : w pokoju z serem musisz wyłapać wszystkie myszy , w bibliotece należy wejść na regał i zabrać trzy wazony z kwiatami . W pokoju z akwarium należy wskoczyć do akwarium i zjeść wszystkie ryby. Uwaga na pływające węże elektryczne - kontakt z nimi powoduje śmierć kota . Jeżeli trafisz do pokoju z klatką , zepchnij ją ze stolika ,a następnie złap uwolnionego w ten sposób kanarka . Możesz się również znaleźć w pokoju z miseczkami napełnionymi mlekiem . Masz teraz okazję napić się do syta , lecz pod jednym warunkiem : że będziesz uważał, by inne koty, znajdujące się w tym samym pomieszczeniu, nie zaatakowały ciebie . Zrobią to , gdy zbyt natarczywie będziesz wypijał mleko z ich miseczek .Pamiętaj także , że jako prawdziwy kot, szukasz również Felicji - wybranki twojego serca . Pokazuje się ona czasem w którymś z okien . Wykorzystaj to i wskocz do tego okna - naprawdę warto . Uważaj na braci broniących dostępu do twej miłości oraz strzelające z łuku amorki . Gdy miłość zwycięży otrzymasz pokaźny zastrzyk z punktów oraz dodatkowego kota . Z pewnością to nie wszystkie atrakcje tego programu , chociażby dlatego , że w prawie każdym pokoju będziesz musiał uporać się z pracującą non stop miotłą , jadowitymi pająkami i czasami przypominającym o sobie psie . Grać w tą grę możesz długo , a gdy się zmęczysz , naciśnij **SPACE BAR** . program zostanie zatrzymany . Aby go kontynuować , naciśnij ponownie ten sam klawisz . Zdobywane przez ciebie punkty wypisywane są na płocie po prawej stronie . Nad nimi podana jest liczba posiadanych jeszcze kotów . Największą ilość zdobytych w grze punktów komputer podaje na płocie po lewej stronie .

COMMODORE AMIGA 2000 — komputer, który robi karierę



OSTATNI MODEL OLIVETTI PC M28
Szesnastobitowy mikroprocesor 80286 pracujący z częstotliwością zegara 8 MHz. M28 wyposażony jest w twardy dysk o pojemności 20 MB i dwa dyski elastyczne po 1.2 MB. Zgodny z IBM.

