

Jezyki programowania.

Jezyk programowania.

Jezyk programowania jest zbiorem symboli, które wraz z regułami posługiwania się nimi, tworzą metodę zapisywania poleceń dla komputera.

Jezyk maszynowy.

Jezyk maszynowy – bajty zapisane jako kombinacje zer i jedynek. Procesor rozpoznaje jedynie rozkazy w postaci kodów binarnych. Pojedynczy kod nazywa się instrukcją lub rozkazem. Każdy typ procesora ma właściwy dla niego zestaw instrukcji, które potrafi wykonać. Każda instrukcja zaleca wykonanie jednego bardzo prostego zadania. Typowe zadania realizowane instrukcjami procesora to:

- przeniesienie zawartości jednej komórki pamięci do innej,
- wskazanie przez procesor numeru komórki, z której ma być pobrana informacja lub do której ma być zapisana,
- dodanie zawartości dwóch rejestrów procesora,
- przesunięcie układu bitów w rejestrze o jedno miejsce w lewo lub w prawo.

Ze względu na bardo proste funkcje pojedynczej instrukcji, program realizujący jakieś praktyczne zadanie składa się z setek i tysięcy takich instrukcji. Każda z nich to odpowiedni układ 0 i 1 o długości jednego bajta. Napisanie programu za pomocą instrukcji wyrażonych tymi kodami, czyli napisanie go w jezyku maszynowym, jest zadaniem niezwykle czasochłonnym i stosowanym bardzo rzadko. Jedynym chyba przypadkiem jest wprowadzanie programów, uruchamiających komputer, do pamięci stałej. Jest to jednak proces technologiczny i raz wprowadzony program nie może zostać zmieniony przez użytkownika.

ASSEMBLER

Ze względu na ogromną trudność posługiwania się jezykiem maszynowym opracowano metodę pisania programów pozwalającą wyeliminować bezpośrednio stosowanie tego jezyka. Poszczególnym rozkazom, w postaci kodów binarnych, przyporządkowano skróty pochodzące od słów angielskich.

Przykłady:

00001110 IMP

00001011 MOV

00001100 RET

IMP (skocz) – polecenie przeskoku do innego miejsca programu, które musi być wskazane odpowiednim parametrem.

MOV (przesuń) – rozkaz przemieszczenia informacji z jednego miejsca do innego. Pełny rozkaz musi jeszcze zawierać źródło i miejsce przeznaczenia przenoszonej informacji.

RET (wróć) – rozkaz zakończenia wykonywania programu i powrót do tego miejsca w programie, z którego nastąpiło poprzednio przejście do wykonywania owego programu.

Zestaw skrótów reprezentujących wszystkie rozkazy rozpoznawane przez procesor, wraz z regułami posługiwania się nimi, tworzy język ułatwiający napisanie programu – ASSAMBLER.

Program napisany w języku ASSAMBLER nie jest zrozumiały dla procesora. Musi najpierw zostać przetransformowany (przetłumaczony) na język maszynowy (kody binarne) i dopiero w tej postaci może być przekazany do wykonania przez komputer. Tłumaczenie programu napisanego w języku ASSAMBLER na język maszynowy wykonuje specjalnie w tym celu napisany program, zwany również ASSAMBLER.

Języki wyższego poziomu.

Idea języka programowania wyższego poziomu.

Przypuśćmy, że chcemy napisać program, który umieści w pamięci komputera dwie liczby, doda je do siebie i wynik umieści w komórce nr zzz. W języku maszynowym, czy w języku ASSAMBLER, program taki musi zawierać instrukcje:

umieść liczbę 35 w komórce nr xxx,
umieść liczbę 22 w komórce nr yyy,
obierz do rejestru A liczbę z komórki nr xxx,
pobierz do rejestru B liczbę z komórki nr yyy,
dodaj zawartość rejestrów A i B, wynik zapisz w A,
prześlij zawartość rejestru A do komórki nr zzz.

W zapisie zbliżonym do arytmetycznego, program taki może mieć postać:

$x \leftarrow 35$

$y \leftarrow 22$

$z \leftarrow x + y$

co należy odczytać tak:

do zmiennej o nazwie x przypisz liczbę 35,

do zmiennej o nazwie y przypisz liczbę 22,

dodaj wartość zmiennej x do wartości zmiennej y i wynik zapisz w zmiennej z.

Tak napisany program może być wykonany przez komputer dopiero po jego przetłumaczeniu na język maszynowy. Schemat logiczny takiego tłumaczenia jest następujący:

$x \leftarrow 35$ weź numer pierwszej wolnej komórki pamięci, przypisz do tego numeru nazwę x i wprowadź do tej komórki liczbę 35,

$y \leftarrow 22$ weź numer następnej wolnej komórki pamięci, przypisz do niego nazwę y i wprowadź do tej komórki liczbę 22,

$z \leftarrow x + y$ pobierz z komórki o numerze przypisanym do nazwy x jej zawartość i przenieś do rejestru A,

pobierz z komórki o numerze przypisanym do nazwy y jej zawartość i przenieś do rejestru B,

dodaj zawartości rejestrów A i B, wynik umieszczając w A,

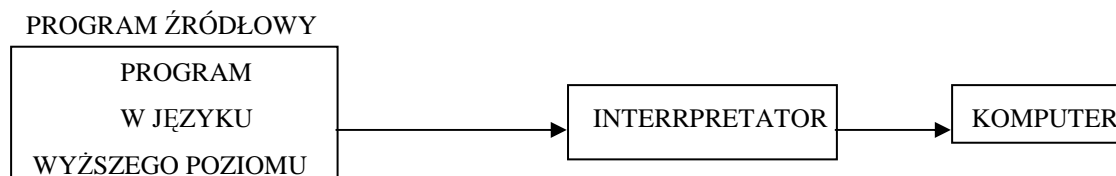
przypisz następnej wolnej komórce pamięci nazwę z i przenieś do tej komórki zawartość rejestru A.

Zestaw umownych skrótów i symboli, wraz z regułami posługiwania się nimi, tworzy język programowania wyższego poziomu, za pomocą którego zapisujemy pierwotną postać programu. Program zapisany w ten sposób nazywa się programem źródłowym. Program taki musi zostać następnie przetworzony na odpowiednie instrukcje w języku maszynowym.

Program, który potrafi zmienić owe umowne symbole i skróty języka wyższego poziomu na ciąg instrukcji języka maszynowego, które, przekazane do procesora, zrealizują zadanie zapisane w programie źródłowym, nazywa się interpretatorem lub kompilatorem (czyli tłumaczem).

Programy interpretujące (interpretatory)

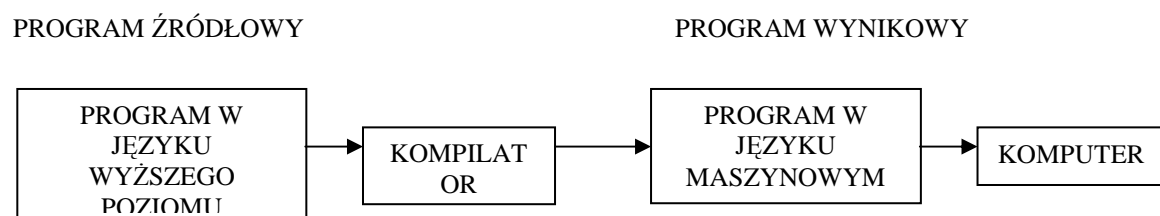
Kolejne instrukcje zapisane w języku wyższego poziomu są tłumaczone na język maszynowy i przekazywane zaraz do wykonania przez procesor.



Wykonanie programu wymaga, aby w pamięci komputera znajdowały się jednocześnie: program źródłowy i interpretator.

Programy kompilujące (kompilatory)

Cały program źródłowy tłumaczony jest na język maszynowy i wynik tłumaczenia zapisany jest na dysku. Ten wynik tłumaczenia jest programem zapisanym w języku maszynowym i nazywa się programem wynikowym.



Program wynikowy może być wielokrotnie wykonywany przez komputer, już bez potrzeby korzystania z programu źródłowego i kompilatora. Programy wynikowe, tworzone przez kompilator, zapisywane są w zbiorach o nazwie identycznej z nazwą programu źródłowego, lecz otrzymują rozszerzenia EXE, niezależnie od rodzaju języka użytego do pisania programu źródłowego.

Interpretacja a kompilacja.

Wykonanie programu w systemie interpretacji, czyli tłumaczenia na bieżąco instrukcji języka wyższego poziomu na język maszynowy, trwa zdecydowanie dłużej niż wykonanie tego samego programu źródłowego, po jego przekompilowaniu. W przypadku korzystania z interpretatora dozwolona wielkość programu źródłowego jest znacznie mniejsza niż w przypadku programu przeznaczonego do kompilacji.

Elementy języka programowania

Słowa kluczowe.

Słowa kluczowe są zastrzeżone. Wolno ich używać tylko w ustalonych znaczeniach, nie wolno zatem używać ich jako identyfikatorów. Oto słowa kluczowe języka Pascal:

<i>and</i>	<i>do</i>	<i>function</i>	<i>nil</i>	<i>program</i>	<i>type</i>
<i>array</i>	<i>downto</i>	<i>goto</i>	<i>not</i>	<i>record</i>	<i>until</i>
<i>begin</i>	<i>else</i>	<i>if</i>	<i>of</i>	<i>repeat</i>	<i>var</i>
<i>case</i>	<i>end</i>	<i>in</i>	<i>or</i>	<i>set</i>	<i>while</i>
<i>const</i>	<i>file</i>	<i>label</i>	<i>packed</i>	<i>then</i>	<i>with</i>
<i>div</i>	<i>for</i>	<i>mod</i>	<i>procedure</i>	<i>to</i>	

Pascal nie odróżnia małych i dużych liter, można więc słowa kluczowe pisać dowolnie, np. Var lub VAR zamiast var.

Identyfikator

Identyfikator służy jako nazwa stałej, zmiennej, typu, procedury, funkcji, programu lub pola w rekordzie. Identyfikator może składać się z liter, cyfr lub znaku podkreślenia_. Nie może zawierać spacji i znaków narodowych, np. ą, ś, ź.

Przykłady:

AlaMaKota
X17
Indeks
Oto_poprawny_identyfikator
WriteLn
Exit

Deklaracja.

Deklaracja – jawne opisanie jakiegoś obiektu jeszcze przed jego wykorzystaniem w programie. Obiektem tym może być na przykład zmienna: deklaracja obejmuje wówczas podanie jej nazwy oraz typu.

Operatory.

Operatory w Pascalu dzielą się na arytmetyczne, logiczne, łańcuchowe, relacyjne i mnogościowe.

1. Operatory arytmetyczne

A) Operatory binarne

Do operatorów arytmetycznych binarnych należą: operator dodawania $+$, odejmowania $-$, mnożenia $*$, dzielenia $/$, dzielenia całkowitego *div* i modulo *mod*.

Wynikiem dzielenia $i \text{ div } j$ jest iloraz i/j zaokrąglony w kierunku zera do liczby całkowitej. Np. wynikiem dzielenia $7 \text{ div } 3$ jest 2.

Wynikiem dzielenia $i \text{ mod } j$ jest reszta z ilorazu i/j , czyli $i \text{ mod } j = i - (i \text{ div } j) * j$. Np. wynikiem dzielenia $18 \text{ mod } 5$ jest 3.

B) Operatory unarne

Do operatorów arytmetycznych unarnych należą: operator identyczności $+$ (który nie zmienia) i operator negacji $-$ (który zmienia znak wyrażenia na przeciwny).

2. Operatory logiczne

W Pascalu są trzy operatory logiczne: unarny operator negacji *not* i binarne operatory alternatywy *or* (czyli „lub”) i koniunkcji *and* (czyli „i”).

Należy pamiętać w wyższym priorytecie operatorów logicznych nad relacyjnymi i stosować nawiasy w wyrażeniach typu $(x > 0)$ and $(y > 0)$.

3. Operator łańcuchowy

Jedynym operatorem łańcuchowym jest operator konkatencji $+$. Powoduje on sklejenie dwóch łańcuchów. Argumentami tego operatora mogą być zarówno obiekty typu String jak Char. Wynik działania jest typu String.

4. Operatory mnogościowe

Operatory mnogościowe działają na zbiorach. Istnieją trzy takie operatory: operator sumy zbiorów $+$, różnicy zbiorów $-$ i przecięcia zbiorów (części wspólnej) $*$.

5. Operatory relacji

Operator	Znaczenie	Typ argumentów	Typ wyniku
=	Równy	prosty, zbiorowy, wskazujący, łańcuchowy	Boolean
<>	Nierówny	prosty, zbiorowy, wskazujący, łańcuchowy	Boolean
<	Mniejszy	prosty, łańcuchowy	Boolean
>	Większy	prosty, łańcuchowy	Boolean
<=	Mniejszy, bądź równy	prosty, łańcuchowy	Boolean
>=	Większy, bądź równy	prosty, łańcuchowy	Boolean
<=	Zawiera się	zbiorowy	Boolean
>=	Zawiera się	zbiorowy	Boolean
in	Należy do	lewy argument: porządkowy, prawy argument: zbiorowy	Boolean

Procedury i funkcje.

Procedura lub funkcja jest to fragment programu zapisany w osobnym bloku. Każda procedura i funkcja składa się z nagłówka i następującego po nim bloku procedur lub funkcji, zawierającego odpowiednie deklaracje i instrukcje. Umożliwia to wydzielenie niezależnej funkcjonalnej części programu i zastąpienie ciągu instrukcji pojedynczym odwołaniem. Do procedury odwołujemy się za pomocą instrukcji procedury, a do funkcji przez wyrażenie, zawierające odwołanie do niej. Procedurę lub funkcję należy uprzednio zadeklarować.

Etapy tworzenia programu.

Problem do rozwiązania.

Warunkiem podstawowym powstania programu komputerowego jest istnienie problemu czy zagadnienia, które chcemy rozwiązać za pomocą komputera. Oczywiście, musi to być problem, dla którego opłaca się napisanie programu. Jeśli chcemy obliczyć wysokość podatku od kwoty, którą zapłacilibyśmy za samochód, to pisanie programu dla realizacji tego zadania byłoby pozbawione sensu. Jeżeli jednak prowadzimy komis samochodowy, sprowadzamy samochody z zagranicy, płacimy cło, podatek, ubezpieczenie itp., to napisanie programu wykonującego te obliczenia może już być opłacalne.

W przypadku nauki programowania zagadnienia do rozwiązania wybieramy w sposób nieco sztuczny. Problem musi być na tyle skomplikowany, by powstały program nie składał się z dwóch czy trzech instrukcji, ale nie może być na tyle zawiły, by ktoś miał poważne trudności ze zrozumieniem ostatecznej formy programu. Przypuśćmy, że mamy ułożyć 100 równań kwadratowych, które zostaną umieszczone w zbiorze zadań z matematyki dla szkół średnich. Dla każdego rozwiązania musimy podać rozwiązanie, które będzie również umieszczone w tym zbiorze. Dodatkowym warunkiem jest, by pierwiastki tych równań były liczbami wymiernymi. Mamy więc zadanie. Zastanówmy się nad możliwymi drogami jego rozwiązania. Równanie kwadratowe ma ogólną postać: $ax^2 + bx + c = 0$, a jego rozwiązaniem są dwa pierwiastki

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, x_2 = \frac{-b + \sqrt{\Delta}}{2a}, \text{ gdzie } \Delta = b^2 - 4ac.$$

Najprostsza, ale bardzo czasochłonna, metoda zrealizowania zadania, to rozwiązywanie kolejnych równań z przypadkowo wybranymi współczynnikami a , b i c . Jeżeli pierwiastki któregoś z równań okażą się liczbami wymiernymi, to równanie to wpisujemy do redagowanego zbioru zadań. Wykonanie tego zadania, bez użycia komputera, zajęłoby prawdopodobnie kilkanaście godzin, dlatego przeznaczenie jednej nawet godziny na napisanie odpowiedniego programu komputerowego, który następnie zrealizuje nasze zadanie w ciągu kilkudziesięciu sekund, jest na pewno opłacalne.

Algorytm.

Algorytm to przepis, według którego chcemy rozwiązać postawiony problem. Algorytm, jako taki, nie ma nic wspólnego z komputerem, jest to opis metody rozwiązania, bez wnikania, jakimi środkami, czy za pomocą jakich urządzeń, będziemy rozwiązywać problem. Dla postawionego sobie zadania wybraliśmy już metodę postępowania. Zapiszemy ją teraz w punktach uwzględniając specyfikę rozwiązywania równań kwadratowych. Zakładamy, że ułożone równania będą rozwiązywane przez uczniów, bez użycia komputera, czy kalkulatora, dlatego narzucamy warunek, by rozwiązanie nie wymagało operowania bardzo dużymi liczbami. Znajomość matematyki sprowadza ten warunek do niezbyt dużych wartości współczynników a , b i c .

- 1) Wybieramy trzy liczby całkowite z przedziału od -10 do 10 . Zapisujemy je jako współczynniki a , b i c .
- 2) Sprawdzamy, czy wartości współczynników a , b i c nie są powtórzeniem wartości otrzymanych w poprzednich obliczeniach. Jeżeli nie są – kontynuujemy wykonywanie programu. Jeżeli są – wracamy do punktu 1).

- 3) Obliczamy tzw. Wyróżnik równania kwadratowego według wzoru $\Delta = b^2 - 4ac$.
- 4) Sprawdzamy, czy delta spełnia warunek $\Delta \geq 0$. Jeżeli tak – liczymy dalej. Jeżeli nie – wracamy do punktu 1).
- 5) Sprawdzamy, czy pierwiastek kwadratowy z delty jest liczbą całkowitą. Jeżeli tak – pierwiastki równania będą liczbami wymiernymi i liczymy dalej. Jeżeli nie – wracamy do punktu 1).
- 6) Obliczamy x_1 według wzoru $x_1 = (-b - \text{pierwiastek}(\Delta)) / (2 * a)$
- 7) Obliczamy x_2 według wzoru $x_2 = (-b + \text{pierwiastek}(\Delta)) / (2 * a)$
- 8) Zapisujemy wartości a, b i c wzięte do obliczeń oraz wartości pierwiastków x_1 i x_2 .
- 9) Do liczby rozwiązań równań dodajemy 1, $\text{liczbar} = \text{liczbar} + 1$.
- 10) Sprawdzamy, czy liczba rozwiązań równania spełnia warunek $\text{liczbar} < 100$. Jeżeli tak – wracamy do punktu 1). Jeżeli nie – kończymy obliczenia.

Następny krok to zapisanie kolejnych etapów opisanych w algorytmie za pomocą konkretnego języka programowania.

Redagowanie programu

Redagowanie programu zwane jest też kodowaniem. W procesie tym zapisujemy zlecenia określone w algorytmie za pomocą elementów konkretnego języka programowania.

Najbardziej racjonalny program powinien składać się z osobnych modułów dla każdego dającego się wyodrębnić zadania. Taki moduł nazywa się podprogramem i jest wywoływany z programu głównego.

Przyjmijmy zasadę, że program ma być złożony z podprogramów. Taki system pisania programów ma wiele zalet. Dwie najważniejsze z nich to:

- każdy podprogram tworzy zamkniętą całość i możemy go dowolnie przeredagować, nie wprowadzając żadnych zmian do pozostałych podprogramów,
- poszczególne podprogramy mogą być wykorzystane przy tworzeniu innych programów.

Każdy program powinien spełniać kilka podstawowych warunków. Jednym z nich jest łatwość korzystania z niego. Program powinien być tak napisany, by dowolny użytkownik komputera potrafił go uruchomić i skorzystać z niego.

Każdy program powinien być starannie objaśniony.

Bardzo ważnym elementem programu jest wprowadzenie danych i wydruk wyników. Program powinien tak działać, by operator nie miał najmniejszych wątpliwości, co w danej chwili trzeba wpisać za pomocą klawiatury. Program powinien „pytać” o potrzebne mu wielkości w sposób prosty i zrozumiały.

Wydruk wyników to następny element programu, decydujący o jego funkcjonalności. Wyniki powinny być wypisane na ekranie w sposób przejrzysty, z nagłówkami i tytułami. Program, który wyświetla kilka liczb i każe operatorowi domyślać się, co one znaczą, jest programem zupełnie złym.

Program i jego elementy

Program napisany w Turbo Pascalu składa się z nagłówka, bloku i znaku . (kropka). Nagłówek składa się ze słowa *program*, po którym podaje się nazwę programu. Blok składa się z opisu danych i części wykonawczej, a kropka kończy tekst programu.

Typy, stałe i zmienne

Stała (literal) jest nazwą pewnej wartości, która w całym programie nie ulega zmianie. Definicja stałej występuje w początkowej części bloku, zwanej częścią opisową (opisem danych).

Np.

```
const a = 10
```

zdefiniowana stała , w całym programie ma wartość 10.

Zmienne mogą reprezentować różne wartości. Zbiór wartości, jakie może przyjąć dana zmienna nazywa się jej typem. W języku Turbo Pascal istnieje wiele typów predefiniowanych, tj. typów zdefiniowanych w tym języku, których użycie wymaga jedynie podania odpowiedniego identyfikatora.

Np.

```
var x : Integer
```

deklaruje zmienną x jako zmienną typu Integer.

Instrukcje.

W części wykonawczej bloku, którą ograniczają słowa *begin* i *end*, znajdują się instrukcje, które opisują czynności wykonywane w programie. Czynności te dotyczą zwykle danych opisanych w części opisowej bloku, choć mogą też wystąpić czynności nie związane z żadnymi danymi. Poszczególne instrukcje programu oddziela się średnikami.

Instrukcja przypisania – nadaje zmiennej określoną wartość.

Np.

```
x:=10:
```

zmiennej x zostaje przypisana wartość 10.

W instrukcjach przypisania i innych instrukcjach programu mogą występować wyrażenia. Wyrażenie jest to sensowna kombinacja operatorów i operandów (argumentów). Przykładami operatorów są: + (plus – operator dodawania), - (minus – operator odejmowania), * (gwiazdka – operator mnożenia), / (kreska ukośna – operator dzielenia), <, > i = (operatory porównań). Operandami wyrażen mogą być m.in. liczby, stałe i zmienne.

Np. po prawej instrukcji przypisania

```
x:=a+10;
```

występuje wyrażenie, które dodaje liczbę 10 do stałej a.

W programie mogą też występować instrukcje, które uzależniają wykonanie pewnych czynności od spełnienia określonego warunku. Tego typu instrukcje nazywamy instrukcjami warunkowymi.

Np. instrukcja

```
if x<2
    then z:=x+a-10;
```

uzależnia nadanie zmiennej z wartości wyrażenia występującego z prawej strony symbolu przypisania od spełnienia przez zmienną x warunku, że jej aktualna wartość jest mniejsza od 2.

Jeśli w zależności od spełnienia pewnego warunku należy wykonać kilka instrukcji, to instrukcje te można połączyć w jedną instrukcję za pomocą tzw. instrukcji złożonej. Instrukcja złożona rozpoczyna się od słowa *begin*, po którym występuje ciąg instrukcji i kończy słowem *end*.

Np. zamiast pisać

```
if x<2
    then z:=x+a-10;
```

```
if x<2
    then y:=3*x;
```

```
if x<2
    then x:=3*x+2;
```

Można po słowie *then* użyć instrukcji złożonej, w której będą występować trzy instrukcje:

```
if x<2
    then begin
        z:=x+a-10;
        y:=3*x;
        x:=3*x+2
    end;
```

Procedury, funkcje i moduły.

Procedury i funkcje umożliwiają podzielenie części wykonawczej programu na fragmenty zapisane w wyodrębniony sposób w części opisowej bloku.

Różnica pomiędzy procedurą i funkcją polega na tym, że zadaniem procedury jest zwykle obliczenie jednej lub kilku wartości, które są przypisane odpowiednim zmiennym znajdującym się na liście argumentów, a zadaniem funkcji jest obliczenie jednej wartości, która podstawiona jest pod jej nazwę. Opis procedury lub funkcji, które podaje się w części opisowej, rozpoczyna się od nagłówka, po którym występuje blok zakończony średnikiem. Nagłówek procedury różni się od nagłówka funkcji. W pierwszym przypadku rozpoczyna się od słowa kluczowego *procedure*, a w drugim od słowa *function*. Ponadto w nagłówku funkcji określa się jej typ.

Np.

```
procedure obliczenia (a,b:real; var s,r,i:real);
```

```
begin
```

```
    s:=a+b;
```

```
    r:=a-b;
```

```
    i:=a*b
```

```
end;
```

Nazwą tej procedury jest obliczenia. Wielkościami, które reprezentują dane wejściowe są a i b, opisane jako wielkości typu Real. Wynikiem wykonania procedury są trzy wielkości: suma a+b, różnica a-b i iloczyn a*b, które przekazuje się na zewnątrz procedury poprzez trzy zmienne s, r oraz i. Wielkości wejściowe (a, b) oraz wyjściowe (s, r, i) nazywa się parametrami formalnymi procedury.

Przypuśćmy, że w programie pod zmiennymi x i y są pamiętane dwie wartości rzeczywiste. Jeśli w części opisowej bloku programu podano definicję procedury obliczenia, to w celu obliczenia sumy, różnicy i iloczynu tych wartości wystarczy wywołać wspomnianą procedurę dla odpowiednich argumentów (parametrów aktualnych). W wywołaniu procedury w miejscu parametru a powinien wystąpić argument x, w miejscu parametru b – argument y, a w miejscu parametrów s, r oraz i – zmienne, np. suma, różnica i iloczyn. Użyte przez nas argumenty powinny być opisane w części opisowej bloku za pomocą konstrukcji

```
var x, y, suma, roznica, iloczyn: Real;
```

Wywołanie procedury dla podanych argumentów będzie miało postać

```
obliczenia (x, y, suma, roznica, iloczyn);
```

Zamiast podanej wyżej procedury obliczenia moglibyśmy zdefiniować trzy funkcje:

```
function suma (a, b: Real) :Real;
```

```
begin
```

```
    suma:=a+b
```

```
end;
```

```
function roznica (a, b: Real) :Real;
```

```
begin
```

```
    roznica:=a-b
```

```
end;
```

```
function iloczyn (a, b: Real) :Real;
```

```
begin
```

```
    iloczyn:=a*b
```

```
end;
```

i zastosować je do obliczenia sumy, różnicy i iloczynu zmiennych x i y. Ponieważ nazwy suma, roznica i iloczyn są teraz identyfikatorami (nazwami) funkcji, więc użycie tych nazw jako nazw zmiennych nie jest możliwe. Możemy jednak użyć dowolnych innych nazw, np. u, v i w, opisując je w programie następująco:

var x, y, u, v, w: Real;

Wywołanie funkcji jest szczególnym przypadkiem wyrażenia i jako takie może wystąpić z prawej strony symbolu := w instrukcji przypisania. Napiżemy zatem:

u:=suma(x, y);

v:=roznica(x, y);

w:=iloczyn(x, y);

Procedury i funkcje można połączyć w większe jednostki programowe zwane modułami. Moduły są pamiętane w oddzielnych zbiorach dyskowych, a ich użycie w określonym programie wymaga tylko ich deklaracji.

Do zadeklarowania jednego modułu służy konstrukcja

uses nazwa;

gdzie nazwa jest jego nazwą. Jeśli deklarujemy kilka modułów, to ich nazwy oddziela się przecinkami.

Struktura prostego programu i jego analiza.

```
program nazwa-programu;  
uses lista-nazw-modułów;  
definicje-i-deklaracje;  
definicje-funkcji-i-procedur;  
begin  
    instrukcje  
end.
```

W różnych programach niektóre z wymienionych elementów mogą nie występować.

Przykład 1.

Program, którego zadaniem jest wypisanie na ekranie tekstu Turbo Pascal.

```
program napis;  
begin  
    writeln ('Turbo Pascal');  
    readln  
end.
```

napis – nazwa programu

begin ... end – część wykonawcza programu

writeln ('Turbo Pascal') – powoduje wyświetlenie na ekranie tekstu podanego w apostrofach i ujętego w nawiasy okrągłe

readln – jest wywołaniem procedury standardowej, na celu ma zatrzymanie przebiegu programu do chwili naciśnięcia na klawiaturze klawisza Enter

Przykład 2.

Program, który po podaniu przez nas dwóch liczb, podaje ich sumę.

```
program suma;  
var a,b,c: integer;  
begin  
    writeln ('Podaj dwie liczby:');  
    readln (a);  
    readln (b);
```

```
c:=a+b;
write ('Suma podanych liczb wynosi ');
writeln(c);
readln
end.
```

var a, b, c: Integer – opis programu

readln (a) – powoduje zatrzymanie przebiegu programu i oczekiwanie na wprowadzenie z klawiatury liczby całkowitej

c:=a+b – instrukcja przypisania, powoduje przypisanie wartości wyrażenia a+b zmiennej c

Różnica pomiędzy procedurą Writeln i Write polega na tym, że po wykonaniu procedury Writeln następuje automatyczne przejście do początku następnego wiersza, a po wykonaniu procedury Write operacja ta nie jest wykonywana.

Przykład 3.

Program drukujący wizytówkę.

```
program wizytówka;
uses Printer;
var imie, nazwisko: string [20];
    ulica, miejscowosc: string [30];
begin
    write ('Imie: ');
    readln (imie);
    write ('Nazwisko: ');
    readln (nazwisko);
    write ('Ulica: ');
    readln (ulica);
    write ('Miejscowosc: ');
    readln (miejscowosc);
    writeln (Lst, imie, ' ', nazwisko);
    writeln (Lst, ulica);
    writeln (Lst, miejscowosc);
    writeln (Lst)
end.
```

Opis programu zawiera deklaracje modułu standardowego Printer, za pomocą którego realizowany jest dostęp do drukarki i deklaracje tzw. zmiennych łańcuchowych imie, nazwisko, ulica i miejscowosc. Lst – identyfikator predefiniowany, powoduje skierowanie wyjścia na drukarkę.

Jeśli na pojawiające się na ekranie żądania wprowadzimy teksty:

Imie: Jan <Enter>

Nazwisko: Kowalski <Enter>

Ulica: Dworcowa 13 m. 2 <Enter>

Miejscowosc: 61-800 Poznan <Enter>

to wydruk będzie miał postać:

Jan Kowalski

Dworcowa 13 m. 2

61-800 Poznan

Przykład 4.

```
program liczenie;
var d,p,q,s,x,y,z: Real;
procedure obliczenia (a,b:Real;
                    var s,d,p,q:Real);
begin
    s:=a+b;
    d:=a-b;
    p:=a*b;
    q:=a/b;
end;
begin
    writeln ('Podaj trzy liczby rzeczywiste: ');
    write ('x=');
    readln (x);
    write ('y=');
    readln (y);
    write ('z=');
    readln (z);
    if y <>0
        then begin
```

```

    obliczenia (x,y,s,d,p,q);
    writeln ('x+y=',s:8:3, ' x-y=' ,d:8:3);
    writeln ('x*y=',p:8:3, ' x/y=' ,q:8:3);
    end;
if z<>0
    then begin
        obliczenia (y,z,s,d,p,q);
        writeln ('y+z=',s:8:3, ' y-z=' ,d:8:3);
        writeln ('y*z=',p:8:3, ' y/z=' ,q:8:3);
        end;
if x<>0
    then begin
        obliczenia (x,z,s,d,p,q);
        writeln ('z+x=',s:8:3, ' z-x=' ,d:8:3);
        writeln ('z*x=',p:8:3, ' z/x=' ,q:8:3) ;
        end;

readln
end.

```

Zastosowanie procedury znacznie skróciło zapis programu, operacje arytmetyczne są określone tylko raz. W programie niektóre zmienne mają takie same nazwy, jak niektóre parametry procedury. Taka konstrukcja jest dopuszczalna, gdyż zakresem ważności parametrów s, d, p i q procedury obliczenia jest blok tej procedury, a zakresem wartości zmiennych o tych samych nazwach – blok programu z wyjątkiem bloku, w którym występują deklaracje tych samych nazw, a więc z wyjątkiem procedury. Tego typu konstrukcję nazywa się przesłonięciem pierwotnej deklaracji.

Obsługa edytora

Operacje na okienkach

Operacje na okienkach mogą być wykonywane przez:

- wybór odpowiedniego polecenia w menu systemu,
- naciśnięcie klawisza lub klawiszy szybkiego wyboru właściwych dla danego polecenia,
- przesunięcie kursora myszki w określone miejsce i naciśnięcie lewego klawisza myszki.

Operacje wykonywane na okienkach

Operacja	Polecenie z menu	Klawisz(e) szybkiego wyboru	Mysz
Otwarcie okienka edycyjnego	File→Open lub File→New	F3	Wybranie polecenia File→Open lub File→New
Zamknięcie okienka i usunięcie go z ekranu	Window→Close	Alt-F3	Nałożenie kursora myszki na piktogram zamknięcia okna i naciśnięcie klawisza myszki
Uaktywnienie okna	Window→List i wybór okna z listy	Alt-numer (1,2,...,9) lub Alt-O i wybór okna z listy	Przesunięcie kursora myszki w dowolne miejsce uaktywnionego okna i naciśnięcie klawisza myszki
Uaktywnienie okienka następnego	Window→Next	F6	Wybranie polecenia Window→Next
Uaktywnienie okienka poprzedniego	Window→Previous	Alt-F6	Wybranie polecenia Window→Next
Przemieszczanie okna na ekranie	Window→Size Move, po czym naciśnięcie klawiszy →, ←, ↑ i (lub) ↓, a następnie naciśnięcie klawisza Enter	Ctrl-F5 po czym naciśnięcie klawiszy →, ←, ↑ i (lub) ↓, a następnie naciśnięcie klawisza Enter	Usytuowanie kursora myszki w miejscu poprzeczki z tytułem okna, po czym naciśnięcie klawisza myszki i przy jego przytrzymaniu ruch myszka, aż do uzyskania pożądanego

			położenia okna
Zmiana rozmiarów okna	Window→Size Move, po czym naciskanie klawiszy	Ctrl-F5 po czym naciskanie klawiszy Shift→, Shift←, Shift↑, Shift↓, aż do uzyskania pożądanego rozmiaru, a następnie naciśnięcie klawisza Enter	Nałożenie kursora myszki na piktogram zmiany rozmiarów okna, po czym naciśnięcie klawisza myszki i przy jego przytrzymaniu ruch myszką, aż do uzyskania pożądanego rozmiaru okna
Przesunięcie zawartości okna	-----	Klawisze przesuwania kursora	Usytuowanie kursora myszki w dowolnym miejscu poprzeczki przesuwania zawartości okna i naciśnięcie klawisza myszki
Powiększenie (zmniejszenie) okna	Window→Zoom	F5	Nałożenie kursora myszki na piktogram powiększenia okna i naciśnięcie klawisza myszki

Sprowadzanie programu z dysku do edytora.

Klawiatura:

- nacisnąć klawisz F10,
- nacisnąć klawisz F lub Enter,
- nacisnąć klawisz O lub za pomocą klawiatury ↑ lub ↓ przesunąć podświetlenie na napis Open i nacisnąć klawisz Enter,
- jeśli program nie znajduje się w skorowidzu bieżącym, to w podokienku wejściowym Name wpisać pełną nazwę skorowidza, w którym program jest zapisany i nacisnąć klawisz Enter, co spowoduje wyświetlenie w podokienku Files listy z nazwami wszystkich zbiorów zapisanych w tym skorowidzu,
- za pomocą klawiszy ↑, ↓, →, ← przesunąć podświetlenie na nazwę właściwego zbioru i nacisnąć klawisz Enter.

Myszka:

- przesunąć kursor myszki na napis File w menu głównym systemu i nacisnąć klawisz myszki,
- przesunąć kursor myszki na napis Open w menu opcji File i nacisnąć klawisz myszki,

- wpisać pełną nazwę skorowidza, w którym poszukiwany program jest zapisany, przesunąć przycisk myszki na napis Open i nacisnąć klawisz myszki,
- jeśli w podokienku Files znajduje się nazwa poszukiwanego zbioru, to przesunąć na nią kursor myszki i nacisnąć jej klawisz, po czym przesunąć kursor myszki na przycisk Open i ponownie nacisnąć klawisz myszki.

Kompilowanie programu.

Kompilacja Alt-F9

Kompilacja i uruchamianie Ctrl-F9

Aby zobaczyć efekty pracy Alt-F5

Powrót do poprzedniej zawartości ekranu Alt-F5

Klawisze redakcyjne.

Operacje podstawowe:

Przesunięcie kursora o jedną pozycję:

w lewo

← lub Ctrl-S

w prawo

→ lub Ctrl-D

Przesunięcie kursora o jedno słowo:

w lewo

Ctrl-← lub Ctrl-A

w prawo

Ctrl-→ lub Ctrl-F

Przesunięcie kursora o jeden wiersz:

w górę

↑ lub Ctrl-E

w dół

↓ lub Ctrl-X

Przesunięcie tekstu na ekranie:

w górę

Ctrl-W

w dół

Ctrl-Z

Przejdź do strony:

poprzedniej

PgUp lub Ctrl-R

następnej

PgDn lub Ctrl-C

Przesunięcie kursora:

do początku wiersza

Home lub Ctrl-Q-S

do końca wiersza

End lub Ctrl-Q-D

do pierwszego wiersza w okienku

Ctrl-Home lub Ctrl-Q-E

do ostatniego wiersza w okienku

Ctrl-End lub Ctrl-Q-X

do poprzedniego położenia

Ctrl-Q-P

do miejsca wystąpienia ostatniego

Ctrl-Q-W

do następnej pozycji tabulacji	Tab lub Ctrl-I
Dopisywanie i usuwanie tekstu:	
Włączenie (wyłączenie) trybu dopisywania tekstu	Ins lub Ctrl-V
Wstawianie pustego wiersza	Ctrl-N
Usunięcie wiersza	Ctrl-Y
Usunięcie jednego znaku:	
z lewej strony kursora	Backspace lub Ctrl-H
z prawej strony kursora	Del lub Ctrl-G
Usunięcie słowa z prawej strony kursora	Ctrl-T
Operacje na blokach tekstu	
Zaznaczenie początku bloku	Ctrl-K-B
Zaznaczenie końca bloku	Ctrl-K-K
Zaznaczenie pojedynczego słowa	Ctrl-K-T
Zaznaczenie wiersza	Ctrl-K-L
Wydrukowanie bloku	Ctrl-K-P
Skopiowanie bloku	Ctrl-K-C
Przeniesienie bloku	Ctrl-K-V
Usunięcie bloku	Ctrl-K-Y lub Ctrl-Del
Inne operacje redakcyjne	
Wczytanie zbioru z dysku	F3
Zapisanie zbioru na dysku	F2 lub Ctrl-K-S
Uaktywnienie głównego menu	F10 lub Ctrl-K-D
Uaktywnienie menu lokalnego	Alt-F10
Powrót do bieżącego okienka edycyjnego z menu	Esc
Wyświetlenie informacji:	
o edytorze	F1
o elemencie języka	Ctrl-F1
Powiększenie (zmniejszenie) okienka	F5
Zmiana rozmiarów lub przesunięcie okienka	Ctrl-F5

Struktura programu.

Program napisany w języku Turbo Pascal składa się z nagłówka programu, deklaracji modułów, bloku i znaku. (kropka)

Nagłówek programu.

Składa się ze słowa kluczowego *program*, po którym występuje nazwa programu.

Zaleca się po nagłówku zamieszczać komentarz opisujący przeznaczenie programu i podający niezbędne informacje o wprowadzanych danych.

Przykłady:

```
program MatrixMult;
```

```
program Program1;
```

Blok.

Składa się z części opisowej (opisu danych) i części wykonawczej. W części opisowej definiuje się typy, nazwy literałów, podaje się deklaracje etykiet i zmiennych, a także definicje i deklaracje procedur i funkcji. Część wykonawcza jest instrukcją złożoną, zawierającą instrukcje, które będą wykonywane sekwencyjnie. Ograniczają ją słowa kluczowe *begin* i *end*.

Struktura programu.

```
program nazwa-programu;  
deklaracje-modułów;  
część-opisowa;  
begin  
    ciąg-instrukcji  
end.
```

Deklaracje modułów.

Moduły służą głównie do grupowania procedur i funkcji w biblioteki oraz do niezależnego pisania poszczególnych części długich programów przez różnych programistów.

Moduł standardowy *System* jest dostępny automatycznie przez każdy program i dowolne inne moduły. Jego deklaracja jest zbędna.

Deklaracja modułów

```
uses lista-nazw-modułów;
```

Przykłady:

uses Printer, Graph;

uses Crt;

Deklaracje etykiet.

Etykieta umożliwia przekazanie sterowania do tej instrukcji za pomocą instrukcji skoku *goto*. Etykieta składa się z identyfikatora lub ciągu do czterech cyfr.

Deklaracja etykiet

```
label lista-etykiet;
```

Przykłady:

label etykieta;

label alfa, beta, 10, 9999;

Definicje nazw literałów.

Literały mogą być zastąpione przez przypisane im identyfikatory. Każde wystąpienie w programie identyfikatora, któremu przypisano literał, jest równoważne wystąpieniu danego literału. Przypisanie identyfikatorów do literałów następuje w opisie danych za pomocą konstrukcji

```
const sekwencja-definicji-stałych;
```

przy czym każda definicja stałej ma postać

identyfikator = wyrażenie-stałe;

lub

identyfikator : oznaczenie-typu = stała-typowa;

Predefiniowane nazwy literałów (zdefiniowane w module *System*)

Nazwa	Wartość
False	Fałsz
True	Prawda
MaxInt	32767
MaxLongInt	2147483647

Przykłady:

```
const tekst = 'TurboPascal'
```

```
const e = 2,7182818285;
```

```
prawda = True;
```

```
nazwisko = 'Kowalski';
```

Definicje typów.

Każdą zmienną występującą w programie należy zadeklarować, tzn. określić jej nazwę oraz wartości, które może ona przyjmować. Zbiór wartości zmiennej nazywa się typem zmiennej.

Deklaracja typów

```
type sekwencja-definicji-typów;
```

przy czym każdy element sekwencji definicji typów ma postać
identyfikator-typu = opis-typu;

Przykłady:

```
type numer = Integer;
```

```
    dzien = (poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela);
```

```
    tablica = array [1..100] of Real;
```

```
type tekst = Char;
```

```
type a = Integer;
```

```
    b = Real;
```

Deklaracje zmiennych.

```
var sekwencja-deklaracji;
```

przy czym każda deklaracja, występująca w sekwencji deklaracji ma postać
lista-identyfikatorów = opis-typu;

Zakresem zmiennej jest blok, w którym zmienna ta została zadeklarowana. Jeżeli blok ten zawiera inne bloki, w których występują deklaracje zmiennych o takim samym identyfikatorze, to następuje przesłonięcie pierwotnej deklaracji zmiennej.

Przykłady:

```
var i, j, k, l: Integer;
```

```
    a, b: Real;
```

```
    log: Boolean;
```

```
    tab: array[1..50] of Real;
```

Definicje oraz deklaracje funkcji i procedur.

Deklaracje funkcji i procedur są niezbędne tylko w przypadku, gdy uporządkowanie definicji funkcji i (lub) procedur jest takie, że odwołanie do nich nie występują w zasięgu definicji. Sytuacja taka pojawia się na przykład, gdy w definicji jednej procedury występuje wywołanie innej, której definicja występuje poza pierwszą procedurą. Wówczas przed definicją pierwszej procedury należy zadeklarować drugą w następujący sposób:

procedure identyfikator-procedury; *forward*;

Podobnie przypadku funkcji

function identyfikator-funkcji; *forward*;

Przykład:

```
program Prog;
...
procedure Procedura2; forward;
procedure Procedura1;
...
begin
    ...
    Procedura 2;
    ...
end {Procedura 1 };
procedure Procedura2;
...
begin
    ...
    Procedura 1;
    ...
end{Procedura2};
begin
    ...
end {Prog}.
```

Typy danych.

Każda zmienna występująca w programie posiada swoją nazwę (identyfikator) i przyjmuje wartości z określonego zbioru. Zbiór ten nazywa się typem zmiennej.

1. Typy proste

A. Typy porządkowe

(1) Typ wyliczeniowy

stosuje się dla zbiorów o niewielkiej liczbie elementów

Przykłady:

- *type* tydzień = (poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela)
- *type* pora_roku = (wiosna, lato, jesień, zima)

(2) Typy całkowite

są podzbiorem zbioru liczb całkowitych

- typ ShortInt (-128, 127)
- typ Byte (0, 255)
- typ Integer (-32768, 32767)
- typ Word (0, 65535)
- typ LongInt (-2147483648, 2147483647)

Przykłady:

- *type* liczba = Integer;
- *var* i, j, k: liczba;

jest to równoważne deklaracji

- *var* i, j, k: Integer;

(3) Typy logiczne

Boolean

False (Fałsz)

True (Prawda)

Przykład:

- *var* plec :Boolean;

(4) Typ znakowy

Char

Elementami są znaki ASCII

Przykład:

var opcja: Char;

(5) Typy okrojone

służą do ograniczania zakresów wartości dowolnego z dotychczas opisanych typów porządkowych.

Przykłady:

- *type* litera = 'A'..'Z';

(litera jest podzbiorem standardowego typu Char)

- *type* tydzień = (poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela);

dni_robocze = poniedziałek, piątek;

(tydzień jest typem wyliczeniowym, dni_robocze typem okrojonym)

- *type* zakres = 0..100;

B. Typy rzeczywiste

Real ($2,9 \cdot 10^{-39}$; $1,7 \cdot 10^{38}$)

Single – pojedyncza długość ($1,5 \cdot 10^{-45}$; $3,4 \cdot 10^{38}$)

Double – podwójna długość ($5 \cdot 10^{-324}$; $1,7 \cdot 10^{308}$)

Extended -rozszerzona długość ($3,4 \cdot 10^{-4932}$; $1,1 \cdot 10^{4932}$)

2. Typy łańcuchowe

Łańcuchy służą do reprezentowania ciągu znaków, w tym niewidocznego znaku spacji. Elementami typu łańcuchowego są łańcuchy o długości od 0 do długości podanej w definicji typu łańcuchowego.

Przykłady:

- *type* nazwisko = *string* [20];

(nazwisko może zawierać dane będące łańcuchami o maksymalnej długości 20 znaków)

- *const* zakres = 100;

type tekst = *string* [zakres];

(rozmiar typu łańcuchowego tekst został zdefiniowany za pomocą nazwy literału (zakres), który oznacza liczbę 100)

3. Typy strukturalne

Do opisu obiektów złożonych

A. Typy tablicowe

Tablica składa się z ustalonej liczby elementów tego samego typu, zwanego typem składowym. Za pomocą tablic są reprezentowane regularne układy danych, np. wektory i macierze.

Przykłady:

- *type* wektor = *array* [0..50] of Integer;

(definicja określa typ wektora jako typ tablicowy, o elementach typu całkowitego, ponumerowanych od 0 do 50. W zapisie matematycznym typ wektor miałby postać

$$\begin{bmatrix} W_0 \\ W_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ W_{50} \end{bmatrix}$$

- *type* macierz = *array* [1..20] of *array* [1..30] of Real;

(typem składowym, tj. typem każdego z 20 elementów typu tablicowego macierz, ponumerowanych od 1 do 20, jest 30-elementowa tablica liczb rzeczywistych o elementach ponumerowanych od 1 do 30. Elementami typu macierz są więc tablice dwuwymiarowe o 600 elementach typu Real. W zapisie matematycznym typ macierz miałby postać

$$\begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,30} \\ m_{2,1} & m_{2,2} & \dots & m_{2,30} \\ \cdot \\ \cdot \\ \cdot \\ m_{20,1} & m_{20,2} & \dots & m_{20,30} \end{bmatrix}$$

- *type* macierz = *array* [1..20, 1..30] of Real;

(taki sam typ tablicowy jak poprzednio)

- *type* tablica = *array* [Boolean, 1..20, znak] of Char;
- *type* tablica = *array* [Boolean, 1..20] of *array* [znak] of Char;
- *type* tablica = *array* [Boolean] of *array* [1..20, znak] of Char;
- *type* tablica = *array* [Boolean] of *array* [1..20] of *array* [znak] of Char;

(wszystkie powyższe definicje typu tablica określają taki sam typ tablicowy. Pierwszy typ określony jest za pomocą predefiniowanego identyfikatora Boolean, który definiuje typ logiczny o dwu wartościach. Drugi typ indeksu określono za pomocą typu okrojonego. Trzeci typ indeksu określono za pomocą nazwy znak, która wcześniej powinna być zdefiniowana jako identyfikator pewnego typu porządkowego.)

- *type* dzien = (poniedziałek, wtorek, sroda, czwartek, piątek, sobota, niedziela);
var godzina : *array* [1..24] of ShortInt;
tydzien : *array* [1..7] of dzien;

(w przykładzie tym występuje opis dwu zmiennych tablicowych. Zmiennymi tymi są godzina i tydzien. Elementami 24-elementowej tablicy godzina są krótkie liczby całkowite – ShortInt. Natomiast elementami 7-elementowej tablicy tydzien – elementy typu dzien, który wcześniej został określony jako typ wyliczeniowy o wyspecyfikowanych elementach.)

- *type* zakres = 10..20;

`tablica = array [zakres] of Char;`

(typ `tablica` określono tu jako zbiór 11-elementowych tablic o elementach typu znakowego `Char`, przy czym elementy te ponumerowane są od 10 do 20)

B. Typ rekordowy

Rekordem nazywa się złożoną strukturę danych, której składowe, zwane polami, mogą mieć różne charakterystyki.

Definicja typu rekordowego

```
type identyfikator-typu rekord
    lista-deklaracji-pól
end;
```

Przykłady:

- `type data = record`

```
    rok: Integer;
    miesiac: 1..12;
    dzien: 1..31
end;
```

- `type data = record`

```
    rok: Integer;
    miesiac: 1..12;
    dzien: 1..31
end;
```

`posiadanie = (posiada, nie_posiada);`

`personalia = record`

```
    nazwisko: string [20];
    imie1, imie2: string [15];
    nazwisko_panienskie: string [20];
    imie_ojca, imie_matki: string [15];
    data_urodzenia: data;
    miejsce_urodzenia: string [20];
    stan_cywilny: (kawaler, panna, zony, zamężna, wolny);
    miejsce_zamieszkania: string [20];
    case dzieci: posiadanie of
        posiada: (data_urodzenia_dziecka:
            array [1..10] of data);
        nie_posiada: (znak: Char)
```

`end;`

(Zdefiniowana dwa typy rekordowe: data i personalia. W definicji typu personalia wykorzystano definicję typu data, a ponadto zawiera ona deklarację wariantową (case). Występująca w deklaracji warunkowej deklaracja pola wyróżnikowego składa się z nazwy pola wyróżnikowego (dzieci) oraz zdefiniowanego wcześniej identyfikatora typu porządkowego (posiadanie). Zauważmy, że pierwszy z 2-elementowego wykazu wariantów (posiada) ma jedno pole (data_urodzenia_dziecka), ale pole to jest 10-elementową tablicą, której każdy element należy do zdefiniowanego wcześniej typu rekordowego data.)

C. Typ zbiorowy

Jest zbiorem potęgowym danego typu porządkowego, tzn. jest zbiorem wszystkich podzbiorów tego typu, w tym zbioru pustego.

Definicja pojedynczego typu zbiorowego

$type \text{ identyfikator-typu} = set \text{ of typ-porzadkowy};$
--

Liczba elementów typu porządkowego, występującego w definicji i będącego typem bazowym, nie może przekraczać 256, przy czym ich liczby porządkowe muszą należeć do przedziału [0, 255]. Oznacza to, że nie jest dozwolone tu wyspecyfikowane żadnego ze standardowych typów ShortInt, Integer, Word i LongInt. Wartości typu zbiorowego zapisuje się przez podanie w nawiasach kwadratowych listy elementów danego zbioru.

Przykłady:

- $type \text{ dni_pracy} = set \text{ of } (poniedzialek, wtorek, sroda, czwartek, piatek);$
(Elementami typu dni_pracy może być dowolny podzbiór zbioru podanych nazw dni tygodnia, m. in.:
[poniedzialek, czwartek]
[piatek]
[]
[sroda, wtorek, piatek])
- $type \text{ znaki} = set \text{ of Char};$
 $male_litery = ste \text{ of } 'a'..'z';$
(Elementami typu znaki sa dowolne podzbiory zbioru znaków ASCII, a elementami typu male_litery – dowolny podzbiór zbioru małych liter.)

Instrukcje.

Instrukcje proste.

1. Instrukcja przypisania.

Służy do przypisania zmiennej nowej wartości. Ogólna jej postać jest następująca:

odwołanie-do-zmiennej: = wyrażenie;

lub

nazwa-funkcji: = wyrażenie;

Przykłady:

`a := 1;`

`b := 2*c-d / (e+f);`

`war_log := True;`

`tekst := `Turbo` + `` + `Paskal`;`

`x := x+1;`

`warunek := a=b;`

2. Instrukcja skoku.

Jej stosowanie nie jest zalecane, zmniejsza bowiem przejrzystość programu, ogranicza optymalizację kodu wynikowego wykonywanego przez kompilator i utrudnia dowodzenie poprawności programu. Instrukcja ta może być zastąpiona instrukcjami „dopóki” lub „powtarzaj”.

Instrukcja skoku ma postać

`goto etykieta`

3. Instrukcja pusta.

Zapisanie instrukcji pustej nie wymaga użycia żadnego symbolu języka i nie powoduje ona wykonania żadnych czynności. Instrukcję pustą stosuje się w tych kontekstach, w których jest wymagane użycie instrukcji, ale chce się uniknąć wykonania jakiegokolwiek czynności, lub w celu ułatwienia opracowywania programu.

Standardowe wejście – wyjście.

Do wprowadzania danych służą standardowe procedury *Read* i *Readln*, których wywołania mają odpowiednio postać:

Read (lista-zmiennych)

i

Readln (lista-zmiennych)

przy czym poszczególne elementy listy zmiennych oddziela się przecinkami.

Wywołanie procedury *Read* powoduje wprowadzenie ciągu znaków. Wywołanie procedury *Readln* powoduje dodatkowo przejście do początku następnego wiersza. Jeśli procedura *Readln* zostanie wywołana bez listy argumentów, to system będzie czekał na naciśnięcie klawisza Enter.

Do wyprowadzania wyników służą procedury standardowe *Write* i *Writeln*, których wywołania mają postać:

Write (lista-argumentów-wyjściowych)

i

Writeln (lista-argumentów-wyjściowych)

przy czym różnica pomiędzy tymi procedurami polega na tym, że procedura *Writeln*, po wykonaniu tych samych czynności co procedura *Write*, powoduje dodatkowo przejście do następnego wiersza. Przejście do następnego wiersza otrzymuje się przez wywołanie procedury *Writeln* bez listy parametrów. Poszczególne elementy listy argumentów wyjściowych oddziela się przecinkami. Każdy z tych elementów powinien mieć jedną z poniższych postaci:

wyrażenie

wyrażenie :długość

wyrażenie :długość :miejsca-dziesiętne

Pierwsze proste programy.

```
program cytat;
{Program wypisuje na ekranie tresc mysli Moliera}
begin
    writeln ('Doprawdy,');
    writeln ('jak to milo');
    writeln ('cos umiec!');
    writeln ('  Molier. ');
    readln
end.
```

```
program NapiszMojeImie;
{Program po podaniu naszego imienia wpisuje zdanie Nazywasz sie...}
var
    imie: string;
begin
    write ('Podaj swoje imie ');
    readln (imie);
    writeln ('Nazywasz sie ',imie,'!');
    readln
end.
```

```
program sklejanie;
{Program prosi nas o podanie swojego imienia i nazwiska, a nastepnie wpisuje zdanie
Nazywasz sie...}
var
    imie, nazwisko: string[16];
    osobnik: string[33];
begin
    write ('Podaj swoje imie ');
    readln (imie);
    write ('Podaj swoje nazwisko ');
```

```
    readln (nazwisko);
    osobnik := imie + '+' + nazwisko;
    writeln ('Nazywasz sie ', osobnik, '!');
    readln
end.
```

```
program powitanie;
{program po podaniu imion osoby witanej i witajacej oraz liczbie calusow wpisuje zdanie z
zastosowaniem powyzzszych danych}
uses crt;
var
    witana, witajaca: string [20];
    ile: byte;
begin
    write ('Imie osoby witanej : ');
    readln (witana);
    write ('Imie osoby witajacej : ');
    readln (witajaca);
    write ('Ile razy calujesz : ');
    readln (ile);
    writeln ('Czesc ', witana, ', wita cie ', witajaca, '. Caluje cie ', ile, ' razy. ');
    readln
end.
```

```
program imiona;
{Program prosi nas o podanie swojego imienia i wieku, i wypisuje zdania z uzyciem
podanych przez nas danych}
var
    imie: string;
    wiek: integer;
begin
    write ('Jak masz na imie? Napisz i nacišnj ENTER ');
    readln (imie);
    write ('Hej, ', imie, '. Ile masz lat? ');
```

```
    readln (wiek);
    writeln (wiek, ' lat to piekny wiek');
    write ('nacisnij ENTER ');
    readln
end.
```

```
program kilometr;
{Program po podaniu przez nas danych: stanu aktualnego i poprzedniego licznika oraz
zuzycia paliwa oblicz srednie spalanie naszego auta}
var
    StanAktualny, StanPoprzedni, Zuzycie: real;
begin
    write ('Podaj aktualny stan licznika kilometrow (nacisnij ENTER) ');
    readln (StanAktualny);
    write ('Podaj poprzeani stan licznika kilometrow (nacisnij ENTER) ');
    readln (StanPoprzedni);
    write ('Podaj zuzycie paliwa w litrach (nacisnij ENTER) ');
    readln (zuzycie);
    write ('Srednie spalanie wynosi ');
    write (100 * Zuzycie / (StanAktualny - StanPoprzedni):2:2, ' 1/100 km');
    readln
end.
```

```
program droga;
{Program oblicza droge w ruch jednostajnie przyspieszonym}
var
    v0, a, t, s: real;
begin
    write ('Podaj wartosc predkosci poczatkowej v0 ');
    readln (v0);
    write ('Podaj wartosc przyspieszenia a ');
    readln (a);
    write ('Podaj wartosc zmiennej t ');
```

```
    readln (t);
    s:=v0*t+a*t*t/2;
    writeln ('Przebyta droga s = ',s:12:5);
    readln
end.
```

```
program rownanie;
{Program rozwiazuje rownanie kwadratowe}
var
    a, b, c: real;
begin
    writeln ('Podaj kolejno liczby a, b, c oddzielajac je odstepami');
    writeln ('i nacišnj ENTER');
    readln (a, b, c);
    writeln ('Rownanie ',a:6:2, 'x +', b:6:2, '=',c:6:2);
    if a <> 0 then
        writeln ('ma rozwiazanie x = ', (c - b) / a :6:2);
    if (a = 0) and (b <> c) then
        writeln ('nie ma rozwiazania');
    if (a = 0) and (b = c) then
        writeln ('jest zawsze spelnione');
    readln
end.
```

```
program rownanie;
{Program rozwiazuje rownanie liniowe}
uses crt;
var
    a, b, x: real;
    odp: char;
begin
    clrscr;
    repeat
```

```

writeln ('Program rozwiazuje rownanie: ax+b=0 dla podanych a, b. ');
    repeat
        write ('Podaj liczbe rozna od zera a = ');
        readln (a);
        until a<>0;
    write ('Podaj liczbe b = ');
    readln (b);
    x:=-b/a;
    writeln ('Rozwiazanie wynosi x = ',x:2:2);
    write ('Czy chcesz liczyc jeszcze raz T\N ');
    readln (odp);
    until upcase (odp)='N'
end.

```

program rownanie;

{program obliczajacy pierwiastki rownania kwadratowego, wykonywany jest tylko wtedy gdy podane wspolczynniki spelniaja warunek, ze delta jest wieksza od 0, w przeciwnym wypadku sygnalizuje blad}

var

a, b, c : Real;

delta : Real;

x1, x2 : real;

begin

writeln ('Podaj wspolczynniki a, b i c');

readln (a, b, c);

delta := b*b-4*a*c;

x1 := (-b + sqrt(delta))/(2*a);

x2 := (-b - sqrt(delta)) / (2*a);

writeln ('x1 = ', x1:2:2, ' x2 = ', x2:2:2);

readln

end.

program rownanie2;

{program obliczajacy pierwiastki rownania kwadratowego}

```

var
    a, b, c: Real;
    delta: Real;
    x1, x2: Real;
begin
    writeln ('Podaj współczynniki a, b i c');
    readln (a, b, c);
    delta := b*b-4*a*c;
    if delta >= 0 then
        begin
            x1 := (-b + sqrt(delta))/(2*a);
            x2 := (-b - sqrt(delta))/(2*a);
            writeln ('x1 = ', x1:2:2, ' x2= ', x2:2:2);
        end
    else
        writeln ('Brak rozwiązania.');
```

readln

end.

```

program prostokat;
{program po podaniu dlugosci bokow prostokata oblicza jego pole i obwod}
var
    a, b: real;
begin
    write ('Dlugosc pierwszego boku = ');
    readln (a);
    write ('Dlugosc drugiego boku = ');
    readln (b);
    writeln ('Pole = ',a*b);
    writeln ('Obwod = ',2*a+2*b);
    readln
end.
```

```

program rzut;
{program po podaniu przez nas predkosci poczatkowej i katu rzutu oblicza maksymalna
wysokosc i zasieg}
uses crt;
var
    v, a:real;
begin
    clrscr;
    writeln ('    v2sin2a');
    writeln ('h = -----');
    writeln ('    2g');
    write ('Podaj predkosc poczatkowa v = ');
    readln (v);
    write ('Podaj kat wyrzutu a = ');
    readln (a);
    writeln ('Maksymalna wysokosc h=',(v*v*sin(a*pi/180)*sin(a*pi/180))/(2*9.81):6:3);
    writeln ('Maksymalny zasieg z =',(v*v/9.81)*sin(2*a*pi/180):6:3);
    readln;
end.

```

```

program kula;
{Porgram po podaniu przez nas promienie kuli oblicza jej pole i objetosc}
uses crt;
var
    r, s, v: real;
begin
    clrscr;
    gotoxy (20,12);
    write ('Podaj promien r = ');
    readln (r);
    s:=4*pi*r*r;
    v:=s*r/3;
    clrscr;
    gotoxy (20,12);

```

```
writeln ('Pole kuli o promieniu r = ',r:5:3,' wynosi s = ',s:5:3);
gotoxy (20,13);
writeln ('Objętość tej kuli wynosi v = ',v:5:3);
readln
end.
```

```
program kula2;
{Program analogiczny jak program kula}
var
    r : real;
begin
    write('Podaj promień r = ');
    readln (r);
    writeln ('Pole kuli o promieniu ',r:8:2,' wynosi S = ',4*pi*r*r:10:4);
    writeln ('Objętość tej kuli wynosi V = ',4*pi*r*r*r/3:10:4);
    readln
end.
```

Proste programy z wykorzystaniem iteracji.

Program TabelaKwadratyPierwiastki;

{Program wyświetla liczby od 1 do 10 i obok kwadraty tych liczb i ich pierwiastki. Dalej dzieje się podobnie, z tym że liczby zmieniają się od 10 do 1}

uses Crt;

var i:Integer;

begin

 clrscr;

 for i:=1 to 10 do

 writeln (i:3, sqr(i):4, sqrt(i):8:3);

 writeln;

 for i:=10 downto 1 do

 writeln (i:3, sqr(i):4, sqrt(i):8:3);

 repeat until keypressed

end.

Program TabelaSinusyCosinusy;

{Program drukuje tabele sinusów i cosinusów kątów od 0 do 90 stopni co 5 stopni}

uses Crt;

var i:Word;

begin

 clrscr;

 writeln ('stopnie sin cos');

 for i:=1 to 18 do

 writeln (5*i:3, sin(5*i*pi/180):12:4, cos(5*i*pi/180):12:4);

 repeat until keypressed

end.

Program LiczbyPierwsze;

{Program drukuje liczby pierwsze od zadanej do zadanej}

uses Crt;

var i,j,a,b,ilosc:Integer;

```

begin
  clrscr;
  write ('Podaj kraniec dolny przedzialu a= ');
  readln (a);
  write ('Podaj kraniec gorny przedzialu b= ');
  readln (b);
  writeln;
  for i:=a to b do
  begin
    ilosc:=0;
    for j:=2 to round(i/2) do
      if i mod j = 0 then ilosc:=ilosc+1;
    if ilosc=0 then write(i:8)
  end;
  repeat until keypressed
end.

```

Program Slinie;

{Program oblicza silnie liczb od 1 do 10}

uses Crt;

var i,j:Byte;

iloczyn:Longint;

begin

clrscr;

writeln;

for j:=1 to 10 do

begin

iloczyn:=1;

for i:=1 to j do iloczyn:=iloczyn*i;

writeln ('Silnia liczby',j:3,' wynosi',iloczyn:10)

end;

repeat until keypressed

end.

```

Program ProstokatZnaczkow;
{Program dtukuje prostokat o bokach X na Y znaczkow zadanych z klawiatury}
uses Crt;
var i,j,x,y:Word;
    znak:Char;
begin
    clrscr;
    write ('Podaj szerokosc prostokata x=');
    readln (x);
    write ('Podaj wysokosc prostokata y=');
    readln (y);
    write ('Podaj rodzaj znaku ');
    readln (znak);
    clrscr;
    for j:=1 to y do
        for i:=1 to x do
            begin
                gotoxy (i,j);
                write (znak);
                delay (10)
            end;
        repeat until keypressed
    end.

```

```

Program TrojkatZaczkow;
{Program drukuje ttojkat o bokach x na y znczkow zadanych z klawiatury}
uses Crt;
var i,j,a,b:Word;
    znak:Char;
begin
    clrscr;
    write ('Podaj wymiar trojkata a=');
    readln (a);
    write ('Podaj rodzaj znaku ');

```

```

readln (znak);
clrscr;
b:=1;
for j:=1 to a do
    begin
        for i:=1 to b do
            begin
                gotoxy (i,j);
                write (znak);
                delay (10)
            end;
            b:=b+1
        end;
    repeat until keypressed
end.

```

Program WczytanieZPetlaWhile;

{Program nie wypusci nas dokad nie wczytamy liczby z przedzialu do 0 do 100. Wersja z petla While.}

```

uses Crt;
var i:Integer;
begin
    clrscr;
    i:=-10; {dla bezpieczenstwa na poczatku warunek niespelniony}
    while (i<0) or (i>100) do
        begin
            write ('Podaj liczbe a>=0 i a<=0 ');
            readln (i);
            if (i<0) or (i>100) then writeln ('Co Ci kazalem!')
            else writeln ('Swietnie Ci poszlo!')
        end;
    repeat until keypressed
end.

```

Program WczytanieZPetlaRepeat;

{Program nie wypusci nas dokad nie wczytamy liczby z przedzialu do 0 do 100. Wersja z petla Repeat. }

uses Crt;

var i:Integer;

begin

clrscr;

i:=-10; {dla bezpieczenstwa na poczatku warunek niespeelniony}

repeat

write ('Podaj liczbe a>=0 i a<=0 ');

readln (i);

if (i<0) or (i>100) then writeln ('Co Ci kazalem!')

else writeln ('Swietnie Ci poszlo!')

until (i>=0) and (i<=100);

repeat until keypressed

end.

Program KtoraCwiartka;

{Program po podaniu przez nas wspolrzednych punktu x i y podaje, która to cwiartka układu wspolrzednych. }

uses Crt;

var x,y:Real;

begin

clrscr;

writeln ('Program poprosi o podanie wspolrzednych punktu');

writeln ('Po wczytaniu program odpowie, która to cwiartka ');

writeln ('plaskiego kartezjanskiego układu wspolrzednych.');

write ('Podaj wspolrzedne punktu x=');

readln (x);

write ('y=');

readln (y);

writeln; writeln; writeln;

if (x=0) and (y=0) then writeln ('Jest to poczatek układu');

if x=0 then if y>0 then

writeln ('Punkt lezy na osi Y miedzy I i II cwiartka')

```

else if y<0 then
    writeln ('Punkt lezy na osi Y miedzy III i IV cwiartka');
if y=0 then if x>0 then
    writeln ('Punkt lezy na osi X miedzy I i Iv cwiartka')
else if x<0 then
    writeln ('Punkt lezy na osi X miedzy II i III cwiartka');
if (x>0) and (y>0) then
    writeln ('Punkt lezy w I cwiartce');
if (x<0) and (y>0) then
    writeln ('Punkt lezy w II cwiartce');
if (x<0) and (y<0) then
    writeln ('Punkt lezy w III cwiartce');
if (x>0) and (y<0) then
    writeln ('Punkt lezy w IV cwiartce');
repeat until keypressed
end.

```

Program PoleTrojkata;

{ Program po podaniu przez nas dlugosci bokow trojkata oblicza jego pole. }

uses Crt,Dos;

var a,b,c,P: real;

begin

clrscr;

writeln ('Program Oblicz Pole Trojkata');

repeat

write ('Podaj bok a=');

readln(a);

until a>0;

repeat

write ('Podaj bok b=');

readln(b);

until b>0;

repeat

write ('Podaj bok c=');

```

        readln(c);
    until c>0;
    P:=a*b*c;
    writeln;
    writeln ('Pole trojkata wynosi ',P );
    repeat until keypressed;
end.

```

Program CzyPodzielne;

{Program po podaniu przez nas dwoch liczb podaje czy pierwsza z nich jest podzielna przez druga. }

```

var A,B:Integer;
begin
    write('A=');
    readln(A);
    write('B=');
    readln(B);
    if B=0 then
        begin
            writeln('Nie wolno dzielic przez 0. ');
            halt;
        end;
    if (A mod B)=0 then Writeln('A jest podzielne przez B. ');
end.

```

program Pola;

{Program pozwala na wybor pole jakiej figury chcemy liczyc, a nastepnie pyta o potrzebne dane i podaje wyniki. }

```

var A,B,H:Real;
    Wybor:Integer;
begin
    writeln('Co chcesz obliczyc ?');
    writeln('1 - Pole prostokata. ');
    writeln('2 - Pole trojkata. ');

```

```

writeln('3 - Pole rombu. ');
write('Wybierz 1,2 albo 3. ');
readln(Wybor);
if Wybor=1 then
begin
    write('Pierwszy bok: A= ');
    readln(A);
    write('Drugi bok: B= ');
    readln(B);
    writeln('Pole prostokata = ',A*B)
end
else
if Wybor=2 then
begin
    write('Podstawa: A= ');
    readln(A);
    write('Wysokosc: H= ');
    readln(H);
    writeln('Pole trojkata = ',0.5*A*H)
end
else
if Wybor=3 then
begin
    write('Bok : A= ');
    readln(A);
    write('Wysokosc: H= ');
    readln(H);
    writeln('Pole rombu = ',A*H)
end
else
writeln('Wolno wybrac tylko 1,2 albo 3. ')
end.

```

Grafika-moduł *Graph*.

Moduł *Graph*, zawierający dużą liczbę procedur i funkcji standardowych, obsługuje grafikę ekranową. Procedury i funkcje tego modułu pozwalają rysować na ekranie pracującym w trybie graficznym krzywe różnych kształtów i kolorów, wypełniać kontury określonym kolorem, a także wyświetlać na ekranie napisy (poziomo i pionowo) w różnych krojach pisma, z możliwością ich zmniejszania i powiększania. Wykorzystywanie standardowych procedur i funkcji modułu *Graph* wymaga jego zadeklarowania za pomocą konstrukcji *uses*.

Ponadto, przed wywołaniem pierwszej funkcji lub procedury graficznej, moduł *Graph* powinien zostać zainicjowany poprzez wywołanie procedury *InitGraph*, która m. in. wczytuje do pamięci i inicjuje odpowiedni sterownik. Na zakończenie wykonywania operacji graficznych powinna być wywołana procedura *CloseGraph*, która powoduje usunięcie z pamięci sterownika i powrót do poprzedniego trybu tekstowego.

Podczas pracy z modułem graficznym istnieje także możliwość przełączenia trybu graficznego na tryb tekstowy i odwrotnie. Służą do tego celu procedury *RestoreCtrMode* i *SetGraphMode*.

Ekran w trybie graficznym posiada inny układ współrzędnych niż w trybie tekstowym. Każdy punkt tego ekranu ma swoje współrzędne, przy czym punkt znajdujący się w lewym górnym narożniku jest oznaczony jako (0, 0). Liczba punktów ekranu wzdłuż osi x i y zależy od monitora i obsługującej go karty graficznej. Na przykład dla karty VGA w trybie wysokiej rozdzielczości punkt w prawym dolnym narożniku ma współrzędne (639, 479).

Przededefiniowane identyfikatory modułu *Graph*.

Stałe oznaczania kolorów i liczby kolorów.

Stałe oznaczania kolorów są wykorzystywane m. in. w procedurach *SetPalette* i *SetAllPalette*. Ich definicja jest następująca (z prawej strony mamy polskie nazwy kolorów):

<i>const Black</i>	=0	-czarny
<i>Blue</i>	=1	-niebieski
<i>Green</i>	=2	-zielony
<i>Cyan</i>	=3	-turkusowy
<i>Red</i>	=4	-czerwony
<i>Magenta</i>	=5	-karmazynowy
<i>Brown</i>	=6	-brązowy
<i>LightGray</i>	=7	-jasnoszary
<i>DarkGray</i>	=8	-ciemnoszary
<i>LightBlue</i>	=9	-jasnoniebieski
<i>LightGreen</i>	=10	-jasnozielony
<i>LightCyan</i>	=11	-jasnoturkusowy

<i>LightRed</i>	=12	-jasnoczerwony
<i>LightMagenta</i>	=13	-jasnokarmazynowy
<i>Yellow</i>	=14	-żółty
<i>White</i>	=15	-biały

Do oznaczania liczby kolorów służy stała *MaxColors* zdefiniowana następująco:

```
const MaxColors = 15;
```

Stałe określenia rodzaju i grubości linii.

Stałe należące do tej grupy służą do oznaczania rodzajów linii (linia ciągła, kropkowana, przerywana, linia symetrii) oraz ich grubości (linia normalna, pogrubiona). Są one stosowane w procedurze *SetLineStyle*, a ich definicja jest następująca (z prawej strony są wyjaśnienia):

Stałe rodzajów linii:		
<i>const SolidLn</i>	=0	- linia ciągła
<i>DottedLn</i>	=1	- linia kropkowana
<i>CenterLn</i>	=2	- linia symetrii
<i>DashedLn</i>	=3	- linia przerywana
<i>UserBitLn</i>	=4	- linia zdefiniowana
Stałe rodzajów grubości:		
<i>NormWidth</i>	=1	- linia o normalnej grubości
<i>ThickWidth</i>	=3	- linia pogrubiona

Stała *UserBitLn* oznacza rodzaj linii zdefiniowany przez programistę (za pomocą procedury *SetLineStyle*).

Stałe definiowania znaków wypełniających kontury.

Stałe te służą do określania sposobu wypełniania konturów i stosowanych w tym celu znaków. Są one wykorzystywane w procedurze *SetFillStyle*, a ich definicja jest następująca:

<i>const EmptyFill</i>	=0
<i>SolidFill</i>	=1
<i>LineFill</i>	=2
<i>LtSlashFill</i>	=3
<i>SlashFill</i>	=4
<i>BkSlashFill</i>	=5
<i>LtBkSlashFill</i>	=6
<i>HatchFill</i>	=7
<i>KhatchFill</i>	=8

InterleaveFill =9
WideDotFill =10
CloseDotFill =11
UserFill =12

Ostatnia stała (*UserFill*) oznacza wypełnianie konturów znakiem zdefiniowanym przez programistę. Znak ten ustala się za pomocą procedury *SetFillPattern*.

Stałe krojów pisma, kierunku wyprowadzania tekstów i rozmiaru liter.

W trybie graficznym dostępnych jest 11 różnych kroju pisma:

- standardowe (ang. Default),
- „triplex”,
- bezszeryfowe (ang. Sansserif),
- gotyckie (ang. Gothik),
- sześć innych krojów ponumerowanych od 5 do 10

o rozmiarach (powiększeniach) od 1 do 10, przy czym poza pierwszym krojem, którego matryca wynosi 8 x 8bitów, wszystkie pozostałe są krojami wektorowymi (poszczególne znaki są zdefiniowane za pomocą wektorów informujących system graficzny o sposobie ich narysowania). Do określenia pierwszych pięciu krojów służą następujące stałe:

const DefaultFont =0
TriplexFont =1
SmallFont =2
SansSerifFont =3
GothikFont =4

Pozostałe kroje są określane tylko za pomocą numerów (od 5 do 10). Podane stałe są wykorzystywane w procedurze *SetTextStyle*.

Z pismem wyprowadzonym na ekran związane są także stałe zdefiniowane następująco (z lewej strony zamieszczamy objaśnienia):

const HorizDir =0 - wyprowadzanie poziome (od strony lewej do prawej)
VertDir =1 -wyprowadzanie pionowe (z góry na dół)
UserCharSize =2 -rozmiar znaku zdefiniowany przez programistę

Stałe justowania tekstu w poziomie i pionie.

Do określenia rodzaju justowania tekstu w poziomie i pionie służą stałe zdefiniowane następująco:

const CenterText =1
LeftText =0
RightText =2

BottomText =0
TopText =2

Stałe te są wykorzystywane w procedurze *SetTextJustify*.

Stałe ograniczania grafiki do bieżącego okna

Stałe te, zdefiniowane następująco:

const ClipOn = *True*
ClipOff = *False*

Służą do określania, czy wyprowadzana grafika ma być ograniczona do bieżącego okna (*ClipOn*) czy też nie (*ClipOff*). Są one wykorzystywane w procedurze *SetViewPort*.

Procedury inicjujące i zamykające tryb graficzny.

Tryb graficzny jest ściśle związany ze sprzętem komputerowym, a dokładniej z kartą graficzną, w jaką jest wyposażony komputer. W Turbo Pascal zainstalowano sterowniki do obsługi różnych kart graficznych. Podczas inicjowania trybu graficznego należy określić właściwy dla danego sprzętu sterownik i tryb pracy ekranu.

Do sprawdzenia sprzętu komputerowego i określenia właściwego dla niego sterownika, jak również graficznego trybu pracy ekranu służy procedura *DetectGraph*, której wywołanie jest następujące:

DetectGraph (*sterownik*, *tryb*)

przy czym oba argumenty wywołania muszą być zmiennymi typu *Integer*.

W wyniku powyższego wywołania zmiennej *sterownik* zostanie przypisana wartość określająca sterownik do obsługi karty graficznej zainstalowanej w zestawie komputerowym, natomiast zmiennej *tryb* – wartość określająca tryb graficzny wysokiej rozdzielczości.

program AM113;

uses *Graph*;

var *sterownik*, *tryb*: *Integer*;

begin

DetectGraph (*sterownik*, *tryb*);

writeln ('sterownik = ', *sterownik*, 'tryb = ', *tryb*)

end.

W celu zastosowania funkcji i procedur modułu *Graph* należy najpierw moduł ten zainicjować poprzez wywołanie procedury *InitGraph* postaci:

InitGraph (*sterownik*, *tryb*, *skorowidz*)

W wywołaniu tym pierwsze dwa argumenty muszą być zmiennymi typu *Integer*, a ostatni - wyrażeniem typu *String*.

Procedury ustalające parametry graficzne.

Procedury opisane w tym punkcie służą do ustalania parametrów graficznych, takich jak: kolory dostępne w palecie barw, kolor rysowania linii, kolor tła oraz rodzaj i sposób wypełnienia zamkniętych konturów. Do ustalenia koloru rysowanych linii i koloru tła służą procedury *SetColor* i *SetBkColor*, których wywołania są następujące:

SetColor (kolor)

oraz

SetBkColor (kolor)

gdzie *kolor* oznacza wielkość typu *Word*, przy czym w ogólności może być wyrażenie. Zakres dopuszczalnych wartości (nie większy niż [0, 15]) zależy od bieżącego sterownika i bieżącego trybu graficznego.

Wywołanie procedury *SetLineStyle* postaci:

SetLineStyle (rodzaj-linii, wzór, grubość)

w którym wszystkie argumenty powinny być wielkościami typu *Word*, ustala rodzaj bieżącej linii i jej grubość. Jako argumenty *rodzaj linii* zaleca się stosowanie następujące predefiniowane stałe: *SolidLn* (linia ciągła), *DottedLn* (linia kropkowana), *CenterLn* (linia symetrii), *DashedLn* (linia przerywana), *UserBitLn* (linia określona przez programistę), a jako argument *grubość* – predefiniowane stałe *NormalWidth* (linia o normalnej grubości) i *ThickWidth* (linia pogrubiona).

Do ustalenia rodzaju i koloru wypełniania zamkniętych konturów, realizowanego przez procedury *Bar*, *Bar3D*, *FillEllipse*, *FillPoly*, *FloodFill*, *PieSlice*, i *Sector* służą procedury *SetFillStyle* i *SetFillPattern*. Wywołanie pierwszej procedury, które ma postać:

SetFillStyle (wzór, kolor)

gdzie oba argumenty powinny być wielkościami typu *Word* (mogą to być wyrażenia), powoduje ustalenie wzoru wypełniania zgodnie z wartością pierwszego argumentu i koloru zgodnie z wartością drugiego argumentu. Jako argument *wzór* zaleca się stosować predefiniowane stałe znaków wypełnienia, a jako argument *kolor* - stałe kolorów. Druga procedura definiująca wypełnienie, o wywołaniu:

SetFillPattern (wzór zdefiniowany, kolor)

pozwala zdefiniować znak wypełniający kontur oraz ustalić dla niego kolor.

Procedury graficzne.

Procedury graficzne są przeznaczone m. in. do rysowania odcinków prostych i krzywych, płaskich figur geometrycznych, a także wypełniania zamkniętych konturów. Rodzaj i kolor rysowanych linii oraz rodzaj i kolor znaków wypełniających muszą być wcześniej ustalone.

Do rysowania odcinków prostych dostępne są module *Graph* trzy procedury:

Line, *LineRel* i *LineTo*, których wywołania są następujące:

Line (*x1*, *y1*, *x2*, *y2*)

LineRel (Δx , Δy)

LineTo (*x*, *y*)

przy czym wszystkie argumenty wywołań powinny być wielkościami typu *Integer* (w ogólności mogą to być wyrażenia).

Wywołanie procedury *Line* powoduje narysowanie odcinka od punktu ekranu o współrzędnych $(x1, y1)$ do punktu o współrzędnych $(x2, y2)$. Aktualnie położenie wskaźnika ekranu nie ulega przy tym zmianie. Za pomocą procedury *LineRel* otrzymuje się odcinek, którego początek znajduje się w miejscu aktualnego położenia wskaźnika ekranu, a koniec jest przesunięty w stosunku do tego punktu o wyspecyfikowaną wielkość Δx w kierunku osi x i wielkość Δy w kierunku osi y . Jeśli więc aktualne położenie wskaźnika ekranu oznaczymy przez (a, b) , to wywołanie procedury *LineRel* spowoduje narysowanie odcinka łączącego punkty (a, b) i $(a+\Delta x, b+\Delta y)$, przy czym wskaźnik ekranu zostanie przesunięty do tego ostatniego punktu. Wywołanie trzeciej procedury, *LineTo*, powoduje narysowanie odcinka od miejsca aktualnego położenia wskaźnika ekranu do punktu o podanych w wywołaniu współrzędnych (x, y) .

Zaznaczenie na ekranie punktu otrzymuje się przez wywołanie procedury *PuttPixel* postaci:

PuttPixel ($x, y, kolor$)

gdzie x i y oznaczają współrzędne punktu w bieżącym oknie i powinny być wielkościami typu *Integer*, a *kolor* oznacza argument typu *Word* określający kolor punktu. Wszystkie argumenty w ogólności mogą być wyrażeniami. Dla oznaczenia koloru zaleca się stosowanie predefiniowanych stałych, przy czym dostępne kolory zależą od bieżącego sterownika i bieżącego trybu graficznego.

Obraz prostokąta i okręgu w bieżącym oknie, narysowany ustalonym rodzajem linii i kolorem, otrzymuje się za pomocą procedur *Rectangle* i *Circle*, których wywołania są następujące:

Rectangle ($x1, y1, x2, y2$)

i

Circle (x, y, r)

gdzie $x1, y1, x2, y2, x$ i y oznaczają wielkości typu *Integer*, a r - wielkość typu *Word*. Wszystkie argumenty mogą być wyrażeniami. Punkt $(x1, y1)$ jest górnym lewym narożnikiem prostokąta, a punkt $(x2, y2)$ - dolnym prawym narożnikiem, przy czym muszą być spełnione warunki:

$0 \leq x1 < x2 \leq GetMaxX$ i $0 \leq y1 < y2 \leq GetMaxY$

Punkt (x, y) jest środkiem okręgu, a wartość argumentu r określa jego promień.

Do rysowania wielokątów służy procedura *DrawPoly*, której wywołanie ma postać:

DrawPoly (*liczba* - punktów, współrzędne punktów)

gdzie *liczba punktów* oznacza dowolną wielkość typu *Word* i określa liczbę wierzchołków wielokąta, a drugi argument oznacza zmienną odpowiadającą parametrowi nieokreślonego typu. Wielokąt jest rysowany linią ustalonego rodzaju i koloru.

Do otrzymania na ekranie łuków okręgów i elips, narysowanych bieżącym rodzajem linii i bieżącym kolorem, służą procedury *Arc* i *Ellipse* o wywołaniach odpowiednio:

Arc (x, y, α, β, r)

i

Ellipse($x, y, \alpha, \beta, xr, yr$)

gdzie x i y oznaczają wielkości typu *Integer*, a $\alpha, \beta, r, xr,$ i yr - typu *Word*. Wszystkie argumenty tych wywołań w ogólności mogą być wyrażeniami. Punkt (x, y) jest środkiem okręgu lub elipsy. Wielkości α i β oznaczają kąt początkowy i kąt końcowy promienia wodzącego, przy czym kąty te są liczone w stopniach (od 0 do 360) względem osi x i w kierunku osi y , tzn. osi poziomej i pionowej.

Pozostałe procedury graficzne powodują oprócz narysowania odpowiedniej figury, także wypełnienie jej wnętrza aktualnie obowiązującym znakiem wypełniającym i kolorem. Są to procedury *Bar, Bar3D, FillEllipse, FloodFill, PieSlice* i *Sector*. Wywołanie procedury *Bar*, które ma postać:

Bar ($x1, y1, x2, y2$)

gdzie wszystkie argumenty oznaczają wielkości typu *Integer*, co powoduje wypełnienie prostokąta o górnym lewym narożniku w punkcie o współrzędnych $(x1, y1)$ i dolnym prawym – $(x2, y2)$. Brzeg prostokąta nie jest zaznaczany, a znaki wypełniające mają postać i kolor określone procedurami *SetFillStyle* lub *SetFillPattern*.

Procedura *Bar3D* pozwala uzyskać na ekranie prostopadłościan, którego przednia ściana jest wypełniona w taki sam sposób, jak poprzednio. Wywołanie procedury ma postać:

Bar3D ($x1, y1, x2, y2, \text{głębokość}, \text{wierzchołek}$)

Pierwsze cztery argumenty mają takie same znaczenie, jak wywołaniu procedury *Bar*. *Głębokość*, która powinna być wielkością typu *Word*, określa liczbę punktów ekranu „głębokość” trójwymiarowości obrazu prostopadłościanu (zwykle przyjmuje się, że *głębokość* jest równa 25% szerokości przedniej ściany prostopadłościanu). Ostatni argument (*wierzchołek*) powinien być wielkością typu *Boolean*. Jeśli jej wartością jest wartość logiczna *True*, to na obrazie prostopadłościanu będzie zaznaczona jego górna płaszczyzna, a gdy wartość logiczna *False*- płaszczyzna górna nie będzie zaznaczona. Widoczne krawędzie prostopadłościanu zaznaczone są bieżącym rodzajem linii i bieżącym kolorem. Procedura *FillPoly* służy do otrzymywania na ekranie wypełnianych wielokątów. Jej wywołanie jest następujące:

FillPoly (*liczba – punktów, współrzędne – punktów*)

gdzie oba argumenty są identyczne, jak w wywołaniu procedury *DrawPoly* i muszą spełniać te same warunki. Procedura *FillPoly* powoduje narysowanie boków wielokąta rodzajem linii i kolorem ustalonymi przez procedury *SetLineStyle* i *SetColor*, przy czym jego wypełnienie znakami i kolorem ustalonymi przez procedurę *SetFillStyle* lub *SetFillPattern*.

Kolejna procedura – *PieSlice* – powoduje narysowanie bieżącym kolorem wycinka kołowego i jego wypełnienie znakami i kolorem zdefiniowanymi za pomocą procedury *SetLineStyle* lub *SetLinePattern*. Wywołanie procedury *PieSlice* ma postać: *PieSlice* (x, y, α, β, r) przy czym argumenty są takie same, jak w wywołaniu procedury *Arc* i spełniają te same warunki.

Wywołanie procedury *FillEllipse* postaci:

FillEllipse ($x, y, \text{półoś-x}, \text{półoś-y}$)

gdzie pierwsze dwa argumenty powinny być typu *Integer*, a ostatnie dwa – typu *Word*, powoduje narysowanie bieżącym kolorem elipsy o środku w punkcie (x, y) oraz półosiach

$półoś-x$ (wzdłuż osi odciętych) i $półoś-y$ (wzdłuż osi rzędnych), po czym wypełnienie tej elipsy znakami wypełnienia o bieżącym wzorze i kolorze.

Rysunek wypełnionego wycinka elipsy otrzymuje się przez wywołanie procedury *Sector* postaci:

Sector ($x, y, \alpha, \beta, półoś-x, półoś-y$)

gdzie pierwsze dwa argumenty powinny być typu *Integer*, a pozostałe – typu *Word*. Kontur wycinka elipsy jest rysowany bieżącym kolorem i wypełniany znakami wypełniania o bieżącym wzorze i kolorze. Kąty są liczone w stopniach w kierunku przeciwnym do ruchu wskazówek zegara, przy czym kąt 0 (stopni) odpowiada ustawieniu małej wskazówki na godzinę trzecią (α oznacza kąt początkowy, β - kąt końcowy).

Do wypełnienia dowolnego zamkniętego konturu służy procedura *FloodFill*, której wywołanie ma postać:

FloodFill ($x, y, brzeg$)

gdzie wszystkie argumenty powinny być typu *Word*. Wielkości x i y oznaczają dowolny punkt zamkniętego konturu, który ma być wypełniony bieżącym wzorcem i kolorem. Wypełnienie nastąpi do brzegu o kolorze określonym przez trzeci argument. Jeśli punkt (x, y) leży na zewnątrz zamkniętego konturu, to bieżącym wzorcem i kolorem zostanie wypełniona część leżąca na zewnątrz konturu.

Funkcje i procedury tekstowe.

Procedury tekstowe modułu *Graph* służą do wykonywania na ekranie napisów podczas pracy w trybie graficznym. Procedury te umożliwiają m. in. wybór kroju pisma, ustalenie pozycji jego wyprowadzenia na ekranie, a także justowanie i centrowanie tekstu.

Do określenia rodzaju pisma, kierunku wyprowadzania (poziomo lub pionowo) oraz jego rozmiaru służy procedura *SetTextStyle*, której wywołanie ma postać:

SetTextStyle (*krój-pisma, kierunek, rozmiar*)

gdzie wszystkie argumenty powinny być wielkościami typu *Word*. Do określenia kroju pisma (o numerze od 0 do 4) i kierunku jego wprowadzania (od lewej do prawej lub z góry na dół) zaleca się stosować predefiniowane stałe. Do określenia rodzaju pisma należy stosować liczby od 1 do 10 podające odpowiednie powiększenia, przy czym dla kroju standardowego (*DefaultPont*) rozmiarowi standardowemu (normalnemu) odpowiada wartość 1, a dla pozostałych krojów (*TriplexPont, SmallPont, SensSerifPont, GothicPont* i o numerach od 5 do 10) – wartość 4.

Tekst wprowadzany na ekranie w trybie graficznym można justować lub centrować względem aktualnego położenia wskaźnika ekranu lub względem określonego punktu. Ustalenia w tym zakresie dokonuje się poprzez wywołanie procedury *SetTextJustify* postaci:

SetTextJustify (*poziom, pion*)

gdzie oba argumenty powinny być typu *Word*. Pierwszy z tych argumentów podaje rodzaj justowania lub określa centrowanie w poziomie, a drugi – w pionie. Zaleca się tu stosować predefiniowane stałe *LeftText, RightText, CenterText, BottomText* i *TopText*.

Do wyprowadzania tekstu na ekran monitora służą procedury *OutText* i *OutTextXY* o wywołaniach:

OutText (*łańcuch*)

i

OutTextXY (*x*, *y* *łańcuch*)

gdzie *łańcuch* oznacza wielkość typu *String*, a *x* i *y* - wielkość typu *Integer*.

Wywołanie procedury *OutText* powoduje wyprowadzenie podanego łańcucha względem aktualnego położenia wskaźnika ekranu. Uwzględnia się przy tym wszystkie ustalenia dokonane za pomocą procedur *SetTextStyle* i *SetTextJustify*. Jeśli wprowadzany łańcuch nie zmieści się na ekranie lub w ustalonym oknie, to zostanie on obcięty. Natomiast wywołanie drugiej procedury powoduje wyprowadzenie tekstu względem punktu o podanych współrzędnych *x* i *y*, także z uwzględnieniem wspomnianych ustaleń.

W wyprowadzaniu tekstów graficznych pomocne są dwie funkcje: *TextHeight* i *TextWidth*, określające rozmiar łańcucha, obie o wartościach typu *Word* i wywołaniach odpowiednio:

TextHeight (*łańcuch*)

i

TextWidth (*łańcuch*)

gdzie *łańcuch* oznacza dowolne wyrażenie typu *String*. Wartością funkcji *TextHeight* jest wysokość łańcucha, a funkcja *TextWidth* – jego szerokość. W obu przypadkach wartości te są podawane za pomocą liczby punktów ekranu, z uwzględnieniem aktualnego kroju pisma i jego rozmiaru.

Inne.

SetGraphMode – przejście do ekranu graficznego

RestoreCrtMode – przejście do ekranu tekstowego

CloseGraph – zamknięcie ekranu graficznego

GetMaxX – najwyższa wartość *x*

GetMaxY – najwyższa wartość *y*

Programy graficzne.

```
program parametry;
{ Program wypisuje na ekranie dwa rozne teksty, każdy w inny sposób. }
uses crt, graph;
var
    ster, tryb: integer;
begin
    ster := detect;
    initgraph (ster, tryb, 'c:\bp\bgi');
    SetTextJustify (RightText, TopText);
    SetTextStyle (TriplexFont, HorizDir, NormWidth);
    PutPixel (450, 100, GetColor);
    OutTextXY (450, 100, 'Co ja jeszcze robie przy tym komputerze !?!');
    SetTextJustify (RightText, TopText);
    SetTextStyle (TriplexFont, VertDir, ThickWidth);
    PutPixel (500, 10, GetColor);
    OutTextXY (500, 10, 'Jest juz pozna noc. ');
    readln;
    CloseGraph;
end.
```

```
program tryb_rysowania;
uses crt, graph;
var
    ster, tryb: integer;
begin
    ster := detect;
    initgraph (ster, tryb, 'c:\bp\bgi');
    SetColor (red);
    SetLineStyle (SolidLn, 0, ThickWidth);
    Rectangle (10, 10, 200, 100);
    SetColor (Yellow);
    SetWriteMode (XORPut);
```

```

    Rectangle (100, 10, 300, 100);
    readln;
    cleardevice;
    SetColor (red);
    SetLineStyle (SolidLn, 0, ThickWidth);
    Rectangle (10, 10, 200, 100);
    SetColor (yellow);
    SetWriteMode (CopyPut);
    Rectangle (100, 10, 300, 100);
    readln;
    CloseGraph;
end.

```

```

program kropka;
uses crt,graph;
var
    sterownik, tryb, x, y, a: Integer;

procedure inicjacja;
begin
    sterownik:=9;
    InitGraph (sterownik, tryb, 'c:\bp\bgi');
end;

begin
    inicjacja;
    SetGraphMode(2);
    randomize ;
    Bar (130, 120, 510, 130);
    Bar (130, 120, 140, 360);
    Bar(140, 350, 510, 360);
    Bar(500, 130, 510, 360);
    Rectangle (140, 130, 500, 350);
    SetViewPort (150, 140, 490, 340, true);

```

```

repeat
    a:=random(16);
    x:=random(400);
    y:=random(360);
    PutPixel (x, y, a);
until keypressed;
repeat until keypressed;
CloseGraph;
end.

```

```

program linia;
uses crt, graph;
var
    sterownik, tryb, x1, y1, x2, y2: Integer;
    a:0..16;

```

```

procedure inicjacja;
begin
    sterownik:=9;
    InitGraph (sterownik, tryb, 'c:\bp\bgi');
end;

```

```

begin
    inicjacja;
    SetGraphMode (2);
    randomize ;
    repeat
        a:=random(16);
        SetColor (a);
        x1:=random(639);
        y1:=random(479);
        x2:=random(639);
        y2:=random(479);
        Line (x1, y1, x2, y2);
    
```

```

        delay(200);
    until keypressed;
    repeat until keypressed;
    CloseGraph;
end.

```

```

program elipsy;
uses crt, graph;
var
    ster, tryb : Integer;
    i, krok, x, y, xsrodek, ysrodek, promien : Word;
begin
    ster:=Detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    xsrodek := GetMaxX div 2;
    ysrodek := GetMaxY div 2;
    promien := 3*(GetMaxY div 5);
    krok := promien div 30;
    GetAspectRatio (x, y);
    for i:=1 to 30 do
        begin
            SetAspectRatio (x, y+i*getMaxX);
            Circle (xsrodek, ysrodek, promien);
            Dec (promien, krok);
            delay (100);
        end;
    Inc (promien, 30*krok);
    for i:=1 to 30 do
        begin
            SetAspectRatio (x+i*GetMaxY, y);
            Dec (promien, krok);
            Circle (xsrodek, ysrodek, promien);
            delay (100);
        end;
    end;
end;

```

```
    readln;
    CloseGraph;
end.
```

```
program walec;
uses crt, graph;
var
    ster, tryb : integer;
begin
    clrscr;
    ster:=Detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    Ellipse (300, 100, 0, 360, 100, 50);
    Ellipse (300, 350, 0, 360, 100, 50);
    MoveTo (200, 100);
    LineRel (0, 250);
    MoveTo (400, 100);
    LineRel (0, 250);
    readln;
    CloseGraph;
end.
```

```
program wypelnianie_kolorem;
uses crt, graph;
var
    znak: Char;
    ster, tryb: Integer;
begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    SetColor (red);
    Arc (160, 100, 0, 180, 50);
    Line (110, 100, 210, 100);
```

```

readln;
SetFillStyle (1, yellow);
FloodFill (180, 99, red);
readln;
SetColor (blue);
SetFillStyle (4, green);
Rectangle (10, GetMaxY div 2, 70, GetMaxX div 2 - 100);
FloodFill (15, 150, blue);
readln;
CloseGraph;
end.

```

```

program przeciecie_okregow_i_prostokatow;
uses crt, graph;
var
    ster, tryb: Integer;
    srodek_x, srodek_y: Integer;
    promien: Integer;
begin
    ster:=detect;
    initgraph (ster, tryb, 'c:\bp\bgi');
    SetWriteMode (XORPut);
    SetLineStyle (SolidLn, 0, ThickWidth);
    Rectangle (20, 20, GetMaxX-20, GetMaxY-20);
    Rectangle (10, 10, GetMaxX div 2, GetMaxY-10);
    Rectangle (GetMaxX div 2 - 30, 30, GetMaxX-30, GetMaxY-30);
    SetLineStyle (SolidLn, 0, NormWidth);
    srodek_x:=GetMaxX div 2;
    srodek_y:=GetMaxY div 2;
    PutPixel (srodek_x, srodek_y, GetColor);
    readln;
    promien:=(GetMaxY-20) div 2;
    Circle (srodek_x, srodek_y, promien);
    readln;

```

```
    Ellipse (srodek_x, srodek_y, 0, 360, promien, promien div 2);
    readln;
    Arc (srodek_x, srodek_y, 0, 180, promien div 2);
    readln;
    CloseGraph;
end.
```

```
program kwadrat;
uses graph;
var
    szer, wys: word;
    x_dl, y_dl: integer;
    ster, tryb: integer;
begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    GetAspectRatio (szer, wys);
    x_dl:=120;
    y_dl:=Round ((szer/wys)*x_dl);
    Rectangle (0, 0, x_dl, y_dl);
    readln;
    CloseGraph;
end.
```

```
program linia;
uses crt, graph;
var
    x_max, y_max: integer;
    i, ster, tryb: integer;
begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    x_max:=GetMaxX;
```

```

y_max:=GetMaxY;
for i:=0 to 3 do
    begin
        SetLineStyle (i, 0, 3);
        Line (0, 10+i*10, x_max, 10+i*10);
    end;
for i:=1 to 20 do
    begin
        SetLineStyle (4, i, 1);
        Line (10+i*25, 80, 10+i*25, y_max-80);
    end;
for i:=0 to 3 do
    begin
        SetLineStyle (4, 256*256-4+i, 3);
        Line (0, y_max-70+i*10, x_max, y_max-70+i*10);
    end;
readln;
CloseGraph;
end.

```

```

program linie;
uses crt, graph;
var
    ster, tryb: integer;

procedure koperta (x1, y1,x2, y2: integer);
begin
    Line (0, 0, (x2-x1) div 2, (y2-y1) div 2);
    Line ((x2-x1) div 2, (y2-y1) div 2, x2-x1, 0);
    Line (0, 0, x2-x1, 0);
    Line (x2-x1, 0, x2-x1, y2-y1);
    Line (0, y2-y1, x2-x1, y2-y1);
    Line (0, 0, 0, y2-y1);
end;

```

```

begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    koperta (10, 10, 300, 200);
    MoveTo (GetMaxX-30, GetMaxY-30);
    PutPixel (GetMaxX,0, GetColor);
    Readln;
    LineRel (-150, -150);
    readln;
    LineTo (GetMaxX-330, GetMaxY-30);
    readln;
    LineRel (-40, -40);
    readln;
    MoveRel (80, 0);
    LineRel (-40, -40);
    readln;
    LineRel (-40, -40);
    CloseGraph;
end.

```

```

program okregi_prostokaty;
uses crt, graph;
var
    ster, tryb: integer;
    srodek_x, srodek_y, promien: integer;
begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    Rectangle (10, 10, GetMaxX-10, GetMaxY-10);
    readln;
    srodek_x:=GetMaxX div 2;
    srodek_y:=GetMaxY div 2;
    PutPixel (srodek_x, srodek_y, GetColor);

```

```

readln;
promien:=(GetMaxY-20) div 2;
Circle (srodek_x, srodek_y, promien);
readln;
Ellipse (srodek_x, srodek_y, 0, 360, promien, promien div 2);
readln;
Arc (srodek_x, srodek_y, 0, 180, promien div 2);
readln;
CloseGraph;
end.

```

```

program slupki;
uses crt, graph;
var
    ster, tryb: integer;
    poziom: integer;
begin
    ster:=detect;
    InitGraph (ster, tryb, 'c:\bp\bgi');
    poziom:=getMaxY div 2 + 50;
    Rectangle (10, poziom, 40, poziom-200);
    readln;
    Bar (45, poziom, 80, poziom-150);
    readln;
    Bar3D (85, poziom, 120, poziom-170, 10, true);
    readln;
    Bar3D (135, poziom, 150, poziom-200, 10, false);
    readln;
    CloseGraph;
end.

```

Instrukcje strukturalne.

1. Instrukcja złożona – jest ciągiem poprzedzonym słowem kluczowym begin i zakończona słowem kluczowym end.

2. Instrukcja warunkowa

A) Instrukcja „jeśli” uzależnia wykonanie innej lub innych instrukcji od spełnienia lub niespełnienia podanego warunku

if wyrażenie *then* instrukcja

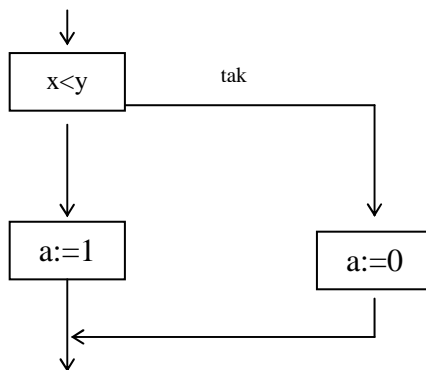
lub

if wyrażenie *then* instrukcja
else instrukcja

Przykłady:

1) *if* $x < y$ *then* $a := 0$

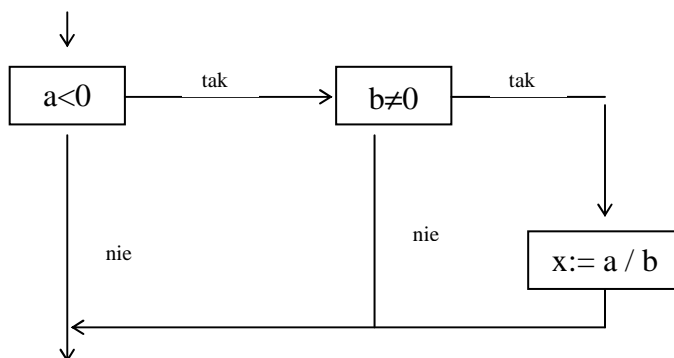
else $a := 1$



2) *if* $a < 0$ *then if* $b \neq 0$ *then* $x := a / b$

lub

if $(a < 0)$ *and* $(b \neq 0)$ *then* $x := a / b$



3) *if a<0 then if b<0 then*

x:= -a-b

else

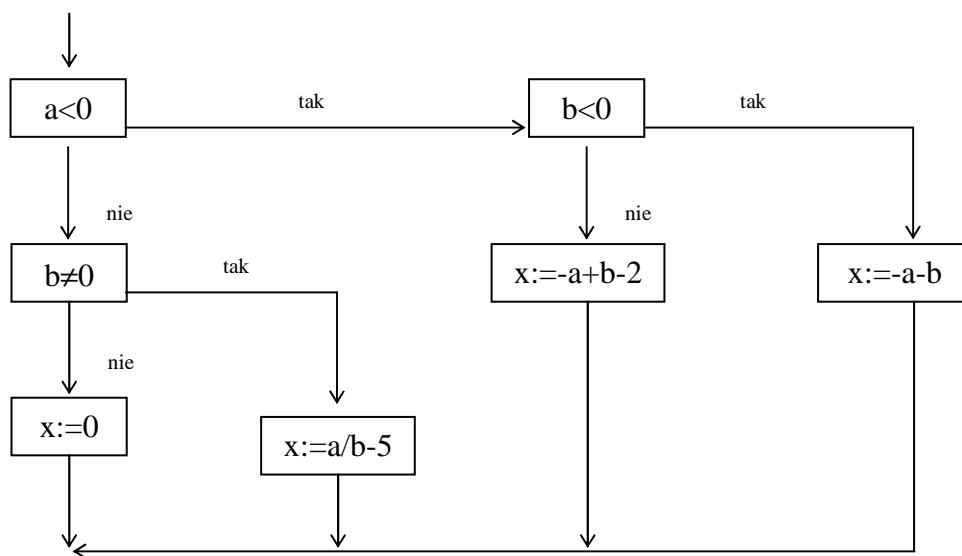
x:= -a+b-2

else if b<>0 then

x:= a/b-5

else

x:= 0



B) Instrukcja wyboru

```
case wyrażenie of
    sekwencja-instrukcji-wyboru
end
```

lub

```
case wyrażenie of
    sekwencja-instrukcji-wyboru
    else instrukcja
end
```

Przykłady:

1) *var* znak: Char;

 k: Integer;

begin

 ...

case znak *of*

 ‘+’: k:=1;

 ‘-’: k:=-1

end;

 ...

end.

W zależności od wartości zmiennej znakowej znak zostanie wykonana instrukcja przypisania k:=1, k:=-1 lub instrukcja pusta.

2) *case* miesiac *of*

 1, 3, 5, 7, 8, 10, 12: dni:=31;

 2: dni:=28;

 4, 6, 9, 11: dni:=30

end

Jeśli założymy, że zmienna miesiac przyjmuje tylko wartości całkowite z przedziału od 1 do 12, a więc zadeklarowana jest następująco

var miesiac: 1..12;

to powyższa instrukcja może być też zapisana w postaci

case miesiac *of*

 2: dni:=28;

 4, 6, 9, 11: dni:=30

else dni:=31

end

3. Instrukcja iteracyjna

A) Instrukcja „dla” – stosuje się w celu wykonania pewnej grupy instrukcji w przypadku, gdy liczba powtórzeń jest znana w dalszym miejscu programu

for zmienna:=wyrażenie1 *to* wyrażenie2 *do* instrukcja

lub

for zmienna:=wyrażenie1 *downto* wyrażenie2 *do* instrukcja

Instrukcja „dla” jest realizowana następująco:

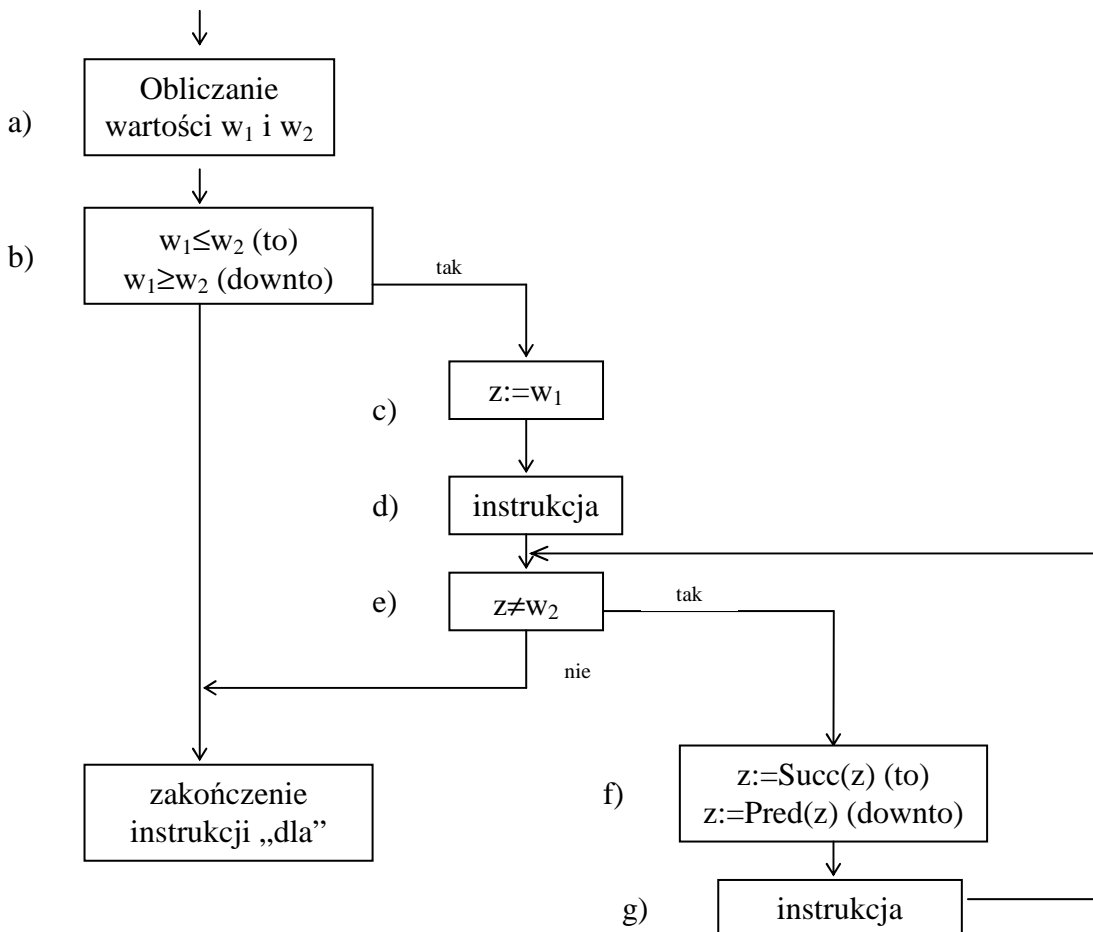
a) oblicza się wartość wyrażen 1 i 2 (w_1 i w_2);

b) sprawdza się, czy wartość w_1 jest

- mniejsza lub równa wartości w_2 , gdy instrukcja „dla” zawiera słowo *to*;
- większa lub równa wartości w_2 , gdy instrukcja zawiera słowo *downto*;

jeżeli warunek ten jest spełniony, to jest wykonywany punkt c), a w przypadku przeciwnym sterowanie przekazywane jest do instrukcji następującej po instrukcji „dla”;

- c) wartość w_1 jest podstawiana pod zmienną sterującą;
- d) wykonuje się instrukcję podaną po słowie kluczowym *do*;
- e) sprawdza się, czy aktualna wartość zmiennej sterującej z w sensie uporządkowania jest różna od wartości w_2 i gdy tak jest, to wykonuje się punkt f), a w przypadku przeciwnym sterowanie jest przekazywane do instrukcji następującej po instrukcji „dla”;
- f) zmiennej sterującej przypisuje się:
 - następną wartość w danym typie, gdy instrukcja „dla” zawiera słowo *to*;
 - poprzednią wartość w danym typie, gdy instrukcja „dla” zawiera słowo *downto*;
- g) wykonuje się instrukcję występującą po słowie kluczowym *do*, po czym przechodzi się do wykonania punktu e).



Przykłady:

- 1) Należy obliczyć sumę S liczb x_1, x_2, \dots, x_n .

```
S:=0
for i:= 1 to n do
    S:= S+x[i];
```

- 2) Należy obliczyć sumę W tych liczb ciągu x_1, x_2, \dots, x_n , które są większe od średniej arytmetycznej z wyrazów ciągu.

```
srednia:=0;
for i:=1 to n do
    srednia:=srednia+x[i];
srednia:=srednia / n;
W:=0;
for i:= 1 to n do
    if x[i] > srednia then
        W:=W+x[i];
```

- 3) Obliczanie sumy wszystkich ujemnych liczb typu *ShortInt*

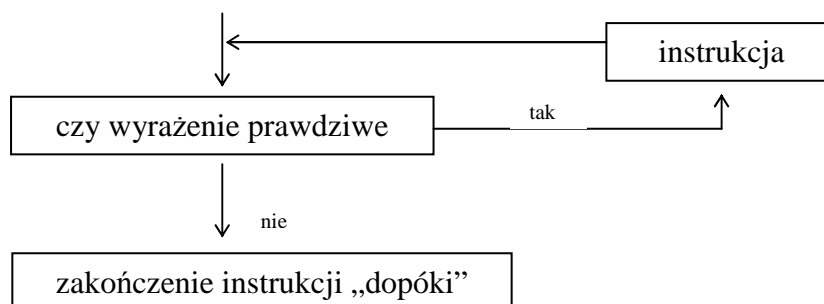
```
suma:=0;
for i:=-128 to -1 do
    suma:=suma+i;
lub
suma:=0;
for i:=-1 downto -128 do
    suma:=suma+i;
```

- 4) Wyświetlenie na ekranie wszystkich małych liter alfabetu łacińskiego.

```
for i:='a' to 'z' do
    write (i);
```

- B) Instrukcja „dopóki” – służy do opisywania iteracji ze sprawdzeniem warunku na początku

```
while wyrażenie do instrukcja
```



Przykłady:

```
1) k:=1;  
   x:=0;  
   while k<10 do  
     begin  
       x:=(x+k)/k;  
       k:=k+1  
     end;
```

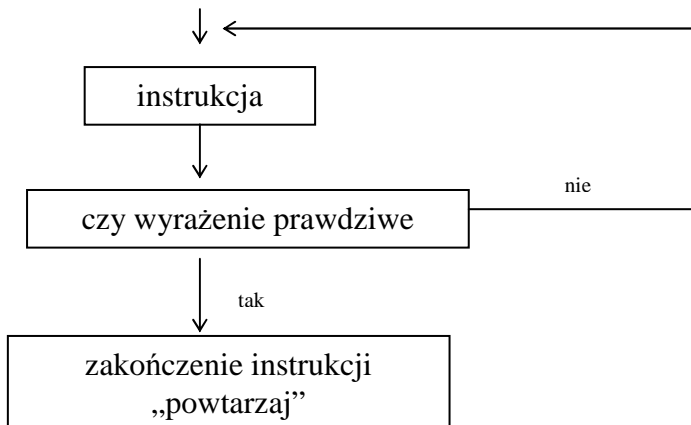
Instrukcja zostanie wykonana 9 razy dla $k = 1, 2, 3, \dots, 9$

```
2) while k>0 do  
   begin  
     k:= k div 2;  
     n:= k*k  
   end;
```

Liczba wykonań iteracji zależy od wartości zmiennej k przed rozpoczęciem jej wykonywania. Jeśli np. początkowa wartość k wynosi 10, to instrukcja zostanie wykonana 4 razy, dla $k = 10, 5, 2$ i 1 .

C) Instrukcja „powtarzaj” – służy do opisywania iteracji ze sprawdzeniem warunku na końcu.

```
repeat  
  instrukcja-1;  
  instrukcja-2;  
  ...  
  instrukcja-n  
until wyrażenie
```



Przykłady:

1) *repeat*

k:= i mod j;

i:= j;

j:= k

until j=0;

Liczba wykonań instrukcji zależy od początkowych wartości zmiennych i oraz j. Jeśli np. zmienne te przed rozpoczęciem wykonania iteracji będą miały wartości odpowiednio 20 i 3, to warunek j=0 zostanie spełniony po trzykrotnym wykonaniu wewnętrznych instrukcji przypisania.

Programy z wykorzystanie instrukcji iteracyjnych.

```
program case1;
uses crt;
var
    a: 0..10;
    zn, odp: char;
begin
    repeat
        clrscr;
        writeln ('Program podana liczbe z przedzialu od 0 do 10 pisze slownie. ');
        write ('Podaj liczbe ');
        readln(a);
        case a of
            0:writeln ('zero');
            1:writeln ('jeden');
            2:writeln ('dwa');
            3:writeln ('trzy');
            4:writeln ('cztery');
            5:writeln ('piec');
            6:writeln ('szesc');
            7:writeln ('siedem');
            8:writeln ('osiem');
            9:writeln ('dziewiec');
            10:writeln ('dziesiec')
        end;
        gotoxy(20,24);
        write('Kontynuowac ? (t/n)');
        readln(odp);
    until upcase(odp)='N';
end.
```

```

program case2;
uses crt;
var
    zn, odp :char;
    a, b:real;

begin
    clrscr;
    repeat
        clrscr;
        gotoxy (1, 5);
        writeln ('Program dla wczytanych z klawiatury liczb a i b oblicza ich ');
        gotoxy (1, 6);
        writeln ('    iloczyn , iloraz , roznice i sume .');
        gotoxy (10, 8);
        write ('Podaj liczbe a : ');
        readln (a);
        gotoxy (10,9);
        write ('Podaj liczbe b : ');
        readln (b);
        gotoxy (12, 12);
        writeln ('i - iloczyn');
        gotoxy (12, 14);
        writeln ('d - iloraz');
        gotoxy (12, 16);
        writeln ('s - suma');
        gotoxy (12, 18);
        writeln ('r - roznica');
        gotoxy (12, 22);
        write ('Wybierz odpowiednia opcje : ');
        readln (zn);
        clrscr;
        case upcase(zn) of
            'I' : writeln ('a*b = ',a*b:4:2);
            'D' : writeln ('a:b = ',a/b:4:2);
        end;
    until zn = 'I' or zn = 'D';
end;

```

```

        'S' : writeln ('a+b = ',a+b:4:2);
        'R' : writeln ('a-b = ',a-b:4:2);
    end;
    gotoxy (20, 24);
    write ('Kontynuowac ? (t/n)');
    readln(odp);
    until upcase(odp)='N';
end.

```

```

program dziel;
uses crt;
var
    liczba,i,j:integer;
    odp:char;
begin
    repeat
        clrscr;
        write ('Podaj liczbe ');
        readln (liczba);
        clrscr;
        writeln ('Dzielnik liczby ',liczba ,' to :');
        j:=0;
        for i:=1 to round(liczba/2) do
            if liczba mod i=0 then
                begin
                    write (i:6);
                    j:=j+1
                end;
        write (liczba:6);
        writeln;
        writeln ('Dzielnikow jest ',j+1);
        writeln;
        write ('Czy chcesz liczyc dalej T/N ');
        readln(odp);
    repeat

```

```
        until upcase(odp)='N';  
end.
```

```
program liczby;  
uses crt;  
var  
    a, b, c: real;  
begin  
    clrscr;  
    writeln ('Program z podanych liczb wybiera najwieksza');  
    write ('Podaj pierwsza liczbe ');  
    readln(a);  
    write ('Podaj druga liczbe ');  
    readln(b);  
    write ('Podaj trzecia liczbe ');  
    readln(c);  
    clrscr;  
    if a>b then  
        begin  
            if a>c then writeln('Najwieksza liczba to ',a:5:3)  
            else writeln('Najwieksza liczba to ',c:5:3);  
        end  
    else  
        begin  
            if b>c then writeln('Najwieksza liczba to ',b:5:3)  
            else writeln('Najwieksza liczba to ',c:5:3);  
        end;  
    readln;  
end.
```

```

program pora;
uses crt;
var
    miesiac, dzien:byte;
    odp:char;
begin
    repeat
        clrscr;
        writeln ('Program po podaniu miesiaca i dnia podaje pore roku');
        write ('Podaj miesiac ');
        readln (miesiac);
        write ('podaj dzien ');
        readln(dzien);
        case miesiac of
            1, 2:writeln ('zima');
            4, 5:writeln ('wiosna');
            7, 8:writeln ('lato');
            10, 11:writeln ('jesien');
            3:if dzien<21 then writeln ('zima') else writeln ('wiosna');
            6:if dzien<24 then writeln ('wiosna') else writeln ('lato');
            9:if dzien<21 then writeln ('lato') else writeln ('jesien');
            12:if dzien<22 then writeln ('jesien') else writeln ('zima')
        end;
        gotoxy (20, 24);
        write ('Kontynuowac ? (t/n)');
        readln (odp);
        until upcase (odp)='N'
    end.

```

```

program pas6;
uses crt, graph;
var
    sterownik, tryb, a, r, s, i, b, c:integer;
procedure inicjacja;

```

```

begin
    sterownik:=9;
    initgraph (sterownik,tryb,'c:\tp\bgi');
end;
begin
    write ('Podaj ilosc okregow a= ');
    readln (a);
    write ('Podaj promien pierwszego r= ');
    readln (r);
    write ('Podaj skok promienia s= ');
    readln(s);
    inicjacja;
    SetGraphMode(2);
    for i:=0 to a-1 do
        begin
            if i mod 2 =0 then
                begin
                    b:=1;
                    c:=1
                end
            else
                begin
                    b:=4;
                    c:=3
                end;
            SetColor (b);
            SetLineStyle (solidln, 0, c);
            Circle(319,239,r);
            r:=r+s
        end;
    repeat until keypressed;
    CloseGraph;
end.

```

```

program pas7;
uses crt;
var
    a: real;
    opcja: char;
begin
    opcja:='n';
    repeat
        clrscr;
        write ('Podaj liczbe: ');
        readln (a);
        writeln ('Kwadrat liczby ',a:5:2,'wynosi: ',sqr(a):5:2);
        writeln('Szescian liczby ',a:5:2,'wynosi: ',sqr(a)*a:5:2);
        if a>=0 then
            begin
                writeln;
                writeln ('Poniewaz nie jest to liczba ujemna oblicze
                pierwiastki');
                writeln ('drugiego i czwartego stopnia.');

```

```
        until opcja in ['n','N']
end.
```

```
program pas8;
uses crt;
var
    i: word;
begin
    clrscr;
    writeln('stopnie sin*sin   cos*cos');
    for i:=1 to 9 do
        writeln(10*i:4,sqr(sin(10*i*pi/180)):12:4,sqr(cos(10*i*pi/180)):12:4);
    repeat until keypressed
end.
```

```
program pas9;
uses crt;
var
    i, j, x, y: word;
    znak: char;
begin
    clrscr;
    write ('Podaj szerokosc prostokata x=');
    readln (x);
    write ('Podaj wysokosc prostokata y=');
    readln (y);
    write ('Podaj rodzaj znaku: ');
    readln (znak);
    clrscr;
    for j:=1 to y do
        for i:=1 to x do
            begin
                gotoxy (i, j);
```

```
                write(znak);
                delay (10)
            end;
        repeat until keypressed
end.
```

```
program suma_for;
uses crt;
var
    i, n: integer;
    S: longint;
    odp: char;
begin
    clrscr;
    writeln ('Program oblicza sume kwadratow liczb od 1 do n. ');
    repeat
        write ('Podaj liczbe n= ');
        readln (n);
        clrscr;
        S:=0;
        for i:=1 to n do
            S:=S+sqr(i);
        writeln ('Suma wynosi S=',S);
        write ('Czy chcesz liczyc jeszcze raz(T\N) ');
        readln (odp);
    until upcase (odp)='N'
end.
```

```
program suma_rep;
uses crt;
var
    i, n: integer;
    S: longInt;
```

```

    odp: char;
begin
    clrscr;
    writeln ('Program oblicza sume kwadratow liczb od 1 do n. ');
    repeat
        write ('Podaj liczbe n= ');
        readln (n);
        clrscr;
        i:=1;
        S:=0;
        repeat
            S:=S+sqr(i);
            i:=i+1;
        until i>n;
        writeln ('Suma wynosi S=',S);
        write ('Czy chcesz liczyc jeszcze raz (T\N) ');
        readln (odp);
    until upcase (odp)='N'
end.

```

```

program suma_while;
uses crt;
var
    i, n: integer;
    S: longInt;
    odp: char;
begin
    clrscr;
    writeln ('Program oblicza sume kwadratow liczb od 1 do n. ');
    repeat
        write ('Podaj liczbe n= ');
        readln (n);
        clrscr;
        i:=1;

```

```

S:=0;
while i<=n do
    begin
        S:=S+sqr(i);
        i:=i+1
    end;
writeln ('Suma wynosi S=',S);
write ('Czy chcesz liczyc jeszcze raz(T\N) ');
readln (odp);
until upcase(odp)='N'
end.

```

```

program TP13;
var
    Suma, LiczbaWprow, Najwieksza: real;
    Licz: integer;
begin
    Suma := 0.0;
    Licz := 0;
    repeat
        WriteLn ('Podaj liczbe nieujemna');
        Write ('(liczba ujemna konczy dzialanie programu) ');
        ReadLn (LiczbaWprow);
        if LiczbaWprow >= 0 then
            begin
                Suma := Suma + LiczbaWprow;
                Licz := Licz + 1;
                if Licz = 1 THEN Najwieksza := LiczbaWprow;
                if LiczbaWprow > Najwieksza then Najwieksza :=
                    LiczbaWprow;
                WriteLn ('Suma ', Licz, ' liczb = ', Suma);
                WriteLn ('najwieksza liczba = ', Najwieksza);
                WriteLn
            end
    end

```

```
        until LiczbaWprow < 0
end.
```

```
program TP14;
var
    liczba, mnoznik: integer;
begin
    Write ('Podaj liczbe calkowita ');
    ReadLn (liczba);
    for mnoznik := 2 to 10 do
        begin
            Write ('Iloczyn ', liczba, ' przez ');
            WriteLn (mnoznik, ' = ', liczba * mnoznik)
        end;
    ReadLn
end.
```

```
program TP15;
var
    i, j, iloczyn: integer;
begin
    for i := 2 to 5 do
        for j := 2 to 7 do
            begin
                iloczyn := i * j;
                WriteLn (i, ' x ', j, ' = ', iloczyn)
            end;
        ReadLn
    end.
```

Programy z procedurami i funkcjami.

```
PROGRAM ciag;
uses crt;
var
a, u, j: integer;

function silnia (liczba:word):real;
var
i:word;
iloczyn:real;
begin
iloczyn:=1;
for i:=1 to liczba do
    iloczyn:=iloczyn*i;
silnia:=iloczyn
end;

procedure wypisz_ciag;
var
j,a:integer;
begin
write('Podaj liczbe '); readln(a);
for j:=0 to a do
    writeln(j:3, ' ',(j+6)/silnia(j) :10:10);
end;

procedure wyraz;
var
u:integer;
begin
write('Podaj liczbe '); readln(u);
writeln(u:3, ' ',(u+6)/silnia(u) :10:10);
end;
```

```

procedure wybor;
var
zn:char;
begin
writeln('Wybierz odpowiednia opcje:');
writeln('1 - oblicza wyraz dla podanej liczby');
writeln('2 - wypisuje caly ciag');
readln(zn);
case zn of
'1': wyraz;
'2': wypisz_ciag;
end;
end;

begin
clrscr;
wybor;
readln
end.

```

```

PROGRAM FunkcjeDwochZmiennych;
{Program zawiera funkcje sin sumy katow i sume sinusow katow.
Kat podaje sie w radianach.}
uses Crt;
var
a, b: Real;

function SinusSumy (alfa, beta:real):real;
begin
SinusSumy:=sin(alfa+beta)
end;

```

```

function SumaSinusow (alfa, beta:real):real;
begin

```

```
SumaSinusow:=sin(alfa)+sin(beta)
```

```
end;
```

```
procedure wczytaj;
```

```
begin
```

```
  clrscr;
```

```
  write ('Podaj kat alfa w radianach ');
```

```
  readln (a);
```

```
  write ('Podaj kat beta w radianach ');
```

```
  readln (b);
```

```
end;
```

```
procedure wypisz;
```

```
begin
```

```
  writeln;
```

```
  writeln ('sin(a+b)=' ,SinusSumy(a,b):8:4);
```

```
  writeln ('sin a + sin b =' ,SumaSinusow(a,b):8:4);
```

```
  repeat until keypressed
```

```
end;
```

```
begin
```

```
  wczytaj;
```

```
  wypisz
```

```
end.
```

```
PROGRAM Lotek;
```

```
{Program praktyczny dla totkowiczow. Wpisac nalezy skreslone liczby  
w 6 zakladach, a potem wylosowane liczby. Program porownuje i podaje  
ile bylo trafnych w kazdym zakladzie.}
```

```
uses Crt;
```

```
type tablica=array [1..7,1..6] of 1..49;
```

```
var
```

```
zaklad:tablica;
```

```
i,j,zakl:1..6;
```

```
trafne:0..6;
```

```
procedure wpisz_zaklady;
```

```
begin
```

```
  clrscr;
```

```
  writeln ('Wpisz teraz liczby w szesciu zakladach');
```

```
  writeln;
```

```
  for i:=1 to 6 do
```

```
    begin
```

```
      writeln('zaklad',i:2);
```

```
      for j:=1 to 6 do
```

```
        begin
```

```
          gotoxy(10*j,i+2);
```

```
          write ('z',j,'=');
```

```
          read(zaklad [i,j])
```

```
        end
```

```
      end
```

```
end;
```

```
procedure wpisz_wyniki;
```

```
begin
```

```
  writeln;
```

```
  writeln ('Wpisz teraz wylosowane liczby');
```

```
  for j:=1 to 6 do
```

```
    begin
```

```
      gotoxy(10*j,12);
```

```
      write ('w',j,'=');
```

```
      read(zaklad [7,j])
```

```
    end;
```

```
end;
```

```
procedure porownaj (nr_zakladu:integer);
```

```
var
```

```
k,l:1..6;
```

```
begin
```

```

trafne:=0;
for k:=1 to 6 do
  for l:=1 to 6 do
    if zaklad[7,1]=zaklad[nr_zakladu,k] then trafne:=trafne+1;
  gotoxy(1,15+nr_zakladu);
  writeln ('W zakladzie',nr_zakladu:2,' trafnych',trafne:2);
end;

```

```

begin
wpisz_zaklady;
wpisz_wyniki;
for zakl:=1 to 6 do
porownaj(zakl);
repeat until keypressed
end.

```

PROGRAM uczniowie;

{Po podaniu danych ucznia i jego ocen okresowych nastepuje obliczenie sredniej ocen}

uses Crt;

var

uczen:record

imie, nazwisko:string[15];

klasa:string[5];

pol,ang,mat,fiz,inf:1..6;

sr:real;

end;

procedure WpiszUcznia;

begin

clrscr;

write ('imie: ');

readln (uczen.imie);

write ('nazwisko: ');

```
readln (uczen.nazwisko);
write ('klasa:      ');
readln (uczen.klasa);
write ('Stopien z j.polskiego      ');
readln (uczen.pol);
write ('Stopien z j.angielskiego      ');
readln (uczen.ang);
write ('Stopien z matematyki      ');
readln (uczen.mat);
write ('Stopien z fizyki      ');
readln (uczen.fiz);
write ('Stopien z informatyki      ');
readln (uczen.inf);
end;
```

```
procedure ObliczSrednia;
var
suma:Byte;
begin
suma:=uczen.pol+uczen.ang+uczen.mat+uczen.fiz+uczen.inf;
uczen.sr:=suma/5
end;
```

```
begin
WpiszUcznia;
ObliczSrednia;
writeln;
write ('Srednia ocen:',uczen.sr:6:3);
repeat until keypressed
end.
```

```

PROGRAM Koch;
{Program rysowania platka Kocha}
uses Crt, Graph;
var
sterownik,tryb,r:Integer;
katAktualny,b:Real;
z:Char;

procedure KL;
begin
while KeyPressed do z:=ReadKey;
repeat until keypressed
end;

procedure PW (katWStopniach:real);
begin
katAktualny:=katAktualny+katWStopniach
end;

procedure LW (katWStopniach:real);
begin
PW (-katWStopniach)
end;

procedure NP (trasa:real);
var
xKonca,yKonca:integer;
begin
xKonca:=round(trasa*cos(katAktualny*pi/180));
yKonca:=round(trasa*sin(katAktualny*pi/180));
LineRel (xKonca,yKonca)
end;

procedure ELEMENT (rzad:integer; bok:real);
begin

```

```

if rzad=0 then NP(bok)
  else
  begin
  ELEMENT (rzad-1,bok/3);
  LW(60);
  ELEMENT (rzad-1,bok/3);
  PW(120);
  ELEMENT (rzad-1,bok/3);
  LW(60);
  ELEMENT (rzad-1,bok/3)
  end
end;

```

```

procedure PLATEK_KOCHA (rzad:integer; bok:real);
var
i:integer;
begin
SetGraphMode(tryb);
SetBkColor(White);
SetColor(Blue);
SetLineStyle(0,0,NormWidth);
MoveTo(100,300);
katAktualny:=270;
PW(30);
for i:=1 to 3 do
  begin
    ELEMENT (rzad,bok);
    PW(120)
  end;
KL;
CloseGraph
end;

```

```

begin
clrscr;

```

```

writeln ('Platek Kocha');
writeln ('-----');
writeln;
write ('rzad fraktala r = ');
readln(r);
write ('bok fraktala b = ');
readln(b);
sterownik:=Detect;
InitGraph (sterownik,tryb, 'c:\bp\bgi');
PLATEK_KOCHA(r,b)
end.

```

```

PROGRAM Koch2;
{Program rysowania platka Kocha - uogolniony dla wielokata}
uses Crt, Graph;
var
sterownik,tryb,n,r:Integer;
katAktualny,b:Real;
z:Char;

procedure KL;
begin
while KeyPressed do z:=ReadKey;
repeat until keypressed
end;

procedure KAT (katWStopniach:real);
begin
katAktualny:=270+katWStopniach
end;

procedure CS;
begin

```

```

SetGraphMode(tryb);
SetBkColor(White);
SetColor(Red);
KAT(0);
SetLineStyle(0,0,NormWidth)
end;

procedure PW (katWStopniach:real);
begin
katAktualny:=katAktualny+katWStopniach
end;

procedure LW (katWStopniach:real);
begin
PW (-katWStopniach)
end;

procedure NP (trasa:real);
var
xKonca,yKonca:integer;
begin
xKonca:=round(1.375*trasa*cos(katAktualny*pi/180));
yKonca:=round(trasa*sin(katAktualny*pi/180));
LineRel (xKonca,yKonca)
end;

procedure ELEMENT (rzad:integer; bok:real);
var
j:integer;
begin
if rzad=0 then NP(bok)
else
begin {do else}
ELEMENT (rzad-1,bok/3); {pierwszy bok elementu fraktala}
LW(180-(360/n)); {obrot przed rysowaniem wielokata}

```

```

for j:=1 to n-1 do {rysowanie wielokata bez ostatniego boku}
begin {do petli rysowania wielokata}
    ELEMENT (rzad-1, bok/3);
    PW(360/n)
end; {do petli rysowania wielokata}
LW(180);
ELEMENT (rzad-1, bok/3)
end {do else}
end;

```

```

procedure PLATEK_KOCHA (rzad:integer; bok:real);
var
i:integer;
begin
CS;
MoveTo(150,300);
for i:=1 to n do
    begin
        ELEMENT (rzad,bok);
        PW(360/n)
    end;
KL;
CloseGraph
end;

```

```

begin
clrscr;
writeln ('Platek Kocha');
writeln ('-----');
writeln;
write ('ile bokow fraktala n = ');
readln (n);
write ('rzad fraktala r = ');
readln(r);
write ('bok fraktala b = ');

```

```
readln(b);
sterownik:=Detect;
InitGraph (sterownik,tryb, 'c:\bp\bgi');
PLATEK_KOCHA(r,b)
end.
```

```
PROGRAM FunkcjeTrygonometryczne;
{Program zawiera zdefiniowane funkcje tg, ctg, kwadraty sin i cos
oraz jedynke trygonometryczna. Na poczatku definiowana jest funkcja
stopnie zmieniajaca kat podany ze stopni na radiany. Dla katow 0 oraz 90
i wielokrotnosci 90 program sie lamie.}
```

```
uses Crt;
var
alfa: Real;
```

```
function stopnie(kat:real):real;
begin
stopnie:=kat*pi/180
end;
```

```
function tg(a:real):real;
begin
tg:=sin(stopnie(a))/cos(stopnie(a))
end;
```

```
function ctg(a:real):real;
begin
ctg:=1/tg(a)
end;
```

```
function sinkwad(a:real):real;
begin
sinkwad:=sqr(sin(stopnie(a)));
end;
```

```
function coskwad(a:real):real;
begin
coskwad:=sqr(cos(stopnie(a)));
end;
```

```
function jedynka(a:real):real;
begin
jedynka:=sinkwad(a)+coskwad(a)
end;
```

```
begin
clrscr;
write ('Podaj kat w stopniach a='); readln (alfa);
writeln ('tg(',alfa:4:0,' stopni)=' ,tg(alfa):8:4);
writeln ('ctg(',alfa:4:0,' stopni)=' ,ctg(alfa):8:4);
writeln ('jedynka(',alfa:4:0,' stopni)=' ,jedynka(alfa):15:10);
repeat until keypressed
end.
```

```
PROGRAM ciagi2;
uses crt;
type wektor=array[0..33] of real;
var
di:wektor;
```

```
function silnia(liczba:word):real;
var
i:word;
iloczyn:real;
begin
iloczyn:=1;
for i:=1 to liczba do
    iloczyn:=iloczyn*i;
```

```
silnia:=iloczyn
```

```
end;
```

```
procedure wypisz_ciag(var tablica:wektor);
```

```
var
```

```
j:word;
```

```
begin
```

```
for j:=0 to 33 do
```

```
begin
```

```
tablica[j]:=(j+6)/silnia(j);
```

```
write(tablica[j]:10);
```

```
end
```

```
end;
```

```
begin
```

```
clrscr;
```

```
wypisz_ciag(di);
```

```
readln
```

```
end.
```

```
PROGRAM objetosci;
```

```
uses crt;
```

```
procedure stozek;
```

```
var
```

```
O,r:real;
```

```
begin
```

```
clrscr;
```

```
write('Podaj promien r= ');readln(r);
```

```
O:=pi*sqr(r)/3;
```

```
writeln;
```

```
writeln('Objetosc wynosi O=',O:5:3);
```

```
writeln;
```

```
write('Wcisnij ENTER');
```

```
readln;  
clrscr  
end;
```

```
procedure kula;  
var  
O,r:real;  
begin  
clrscr;  
write('Podaj promien r= ');readln(r);  
O:=4*pi*sqr(r)*r/3;  
writeln;  
writeln('Objetosc wynosi O=',O:5:3);  
writeln;  
write('Wcisnij ENTER');  
readln;  
clrscr  
end;
```

```
procedure walec;  
var  
O,r,h:real;  
begin  
clrscr;  
write('Podaj promien r= ');readln(r);  
write('Podaj wysokosc h= ');readln(h);  
O:=pi*sqr(r)*h;  
writeln;  
writeln('Objetosc wynosi O=',O:5:3);  
writeln;  
write('Wcisnij ENTER');  
readln;  
clrscr  
end;
```

```

procedure wybor;
var
zn:char;
begin
repeat
gotoxy(12,5);
writeln('Program oblicza objetosci figur obrotowych. ');
gotoxy(12,12);
writeln('k - obj. kuli');
gotoxy(12,14);
writeln('s - obj. stozka');
gotoxy(12,16);
writeln('w - obj. walca');
gotoxy(12,18);
writeln('K - koniec');
gotoxy(12,22);
write('Wybierz odpowiednia opcje : ');
readln(zn);
clrscr;
case zn of
'k' : kula;
's' : stozek;
'w' : walec;
end;
until zn='K';
end;

```

```

procedure haslo;
const sekret = 'PASCAL';
type lancuch8 = string [8];
var koniec:boolean;
function czyt : lancuch8;
var licznik :byte;
    lancuch :lancuch8;
    znak :char;

```

```

begin
licznik:=0;
lancuch:="";
znak:=' ';
repeat
  if keypressed then
    begin
      znak:=upcase(readkey);
      write('*');
      if ord(znak)<>13 then
        lancuch:=lancuch+znak;
      inc(licznik)
    end
until(licznik=8) or (ord(znak)=13);
czyt:=lancuch
end;
begin
textbackground(red);
textcolor(yellow);
window(2,12,79,12);
clrscr;
koniec:=false;
writeln;
write('          Podaj haslo : ');
if czyt<>sekret then
  begin
    writeln;
    write('    Nie podales poprawnego hasla. Sprobuj jeszcze raz : ');
    if czyt<>sekret then
      begin
        writeln;
        write('    Przepraszam, ale nie masz prawa dostepu');
        write(' do programu. ');
        koniec:=true
      end
    end

```

```

end;
if not koniec then
  begin
    writeln;
    write('          Witam w programie!')
  end;
  delay (2000);
  textbackground(0);
  textcolor(15);
  window(1,1,80,25);
  clrscr;
  if koniec then HALT
end;

```

```

begin
  clrscr;
  haslo;
  wybor;
end.

```

```

PROGRAM pas10;
uses crt;
var
  a,b,c:real;

procedure trojmian(a,b,c:real);
var
  x1,x2,delta:real;
begin
  delta:=sqr(b)-4*a*c;
  if delta>0 then
    begin
      x1:=-b-sqrt(delta)/(2*a);writeln('x1=',x1:5:2);
      x2:=-b+sqrt(delta)/(2*a);writeln('x2=',x2:5:2)
    end
  end;
end;

```

```

    end;
if delta=0 then
    begin
        x1:=-b/(2*a);writeln('x=',x1:5:2)
    end;
if delta<0 then
    writeln('Rownanie nie ma pierwiastkow rzeczywistych.')
end;

begin
clrscr;
write('Podaj wspolczynnik a=');readln(a);
write('Podaj wspolczynnik b=');readln(b);
write('Podaj wspolczynnik c=');readln(c);
trojmian(a,b,c);
repeat until keypressed
end.

```

```

PROGRAM pas11;
uses crt;
var
a,b:integer;

procedure liczby_pierwsze(a,b:integer);
var
i,j,ilosc:integer;
begin
for i:=a to b do
    begin
        ilosc:=0;
        for j:=2 to round(i/2) do
            if i mod j =0 then ilosc:=ilosc+1;
        if ilosc=0 then write(i:8)
        end;
    end;
end;

```

```
repeat until keypressed  
end;
```

```
begin  
clrscr;  
write('Podaj kraniec dolny przedzialu a= ');readln(a);  
write('Podaj kraniec gorny przedzialu b= ');readln(b);  
liczby_pierwsze(a,b)  
end.
```

```
PROGRAM pas12;  
uses crt;
```

```
function silniat(liczba:word):longint;  
var  
i:word;  
iloczyn:longint;  
begin  
iloczyn:=1;  
for i:=1 to liczba do  
    iloczyn:=iloczyn*i;  
silniat:=iloczyn  
end;
```

```
function silniarek(liczba:word):longint;  
begin  
if liczba=0 then silniarek:=1  
    else silniarek:=liczba*silniarek(liczba-1)  
end;  
procedure wypiszsilnie;  
var  
j:word;  
begin  
writeln('liczba silnia');
```

```
for j:=1 to 6 do
  writeln(j:3,silniat(j):10)
end;
```

```
begin
clrscr;
wypiszsilnie;
repeat until keypressed
end.
```

```
PROGRAM pas13;
```

```
uses crt;
```

```
var
```

```
a,b,c,x:real;
```

```
function potega(x:real;n:word):real;
```

```
var
```

```
i:word;
```

```
iloczyn:real;
```

```
begin
```

```
iloczyn:=1;
```

```
for i:=1 to n do
```

```
  iloczyn:=iloczyn*x;
```

```
potega:=iloczyn
```

```
end;
```

```
procedure wielomian(a,b,c,x:real);
```

```
begin
```

```
writeln('Wartosc wielomianu wynosi:', a*potega(x,5)+b*potega(x,4)+c*potega(x,3):10:3)
```

```
end;
```

```
begin
```

```
clrscr;
```

```
write('Podaj wspolczynnik a= ');readln(a);
```

```
write('Podaj współczynnik b= ');readln(b);
write('Podaj współczynnik c= ');readln(c);
write('Podaj zmienna x= ');readln(x);
clrscr;
wielomian(a,b,c,x);
repeat until keypressed
end.
```

```
PROGRAM pas14;
uses crt;
type zakupy=record
    nazwa:string[15];
    ilosc:byte;
    cena,zaplata:real
end;
tablica=array[1..3] of zakupy;
var
karta:tablica;

procedure wczytaj(var karta:tablica);
var
i:byte;
begin
for i:=1 to 3 do
    with karta[i] do
        begin
            write('Podaj nazwe towaru: ');readln(nazwa);
            write('Podaj ilosc towaru: ');readln(ilosc);
            write('Podaj cene towaru: ');readln(cena);
            zaplata:=ilosc*cena
        end
    end;

procedure wypisz(karta:tablica);
```

```

var
i:byte;
raz:real;
begin
writeln(' NAZWA   SZT.   CENA   ZAPLATA');
raz:=0;
for i:=1 to 3 do
  with karta[i] do
    begin
      writeln(nazwa:10,' ',ilosc:2,' ',cena:4:2,' ',zaplata:4:2);
      raz:=raz+zaplata
    end;
writeln('-----');
writeln('RAZEM           ',raz:4:2)
end;

begin
clrscr;
wczytaj(karta);
clrscr;
wypisz(karta);
repeat until keypressed
end.

PROGRAM pas15;
uses crt,graph;
var
sterownik ,tryb,x,y,b,r:integer;

procedure inicjacja;
begin
sterownik:=9;
initgraph(sterownik,tryb,'c:\tp\bgi');
end;

```

```
procedure kwadrat_opisany(x,y,b:integer);
begin
rectangle(x-round(0.5*b),y-round(0.5*b),x+round(0.5*b),y+round(0.5*b))
end;

begin
write('Podaj wspolrzeczna srodka okregu x= ');readln(x);
write('Podaj wspolrzeczna srodka okregu y= ');readln(y);
write('Podaj promien okregu r= ');readln(r);
inicjacja;
setgraphmode(2);
setcolor(15);
circle(x,y,r);
kwadrat_opisany(x,y,2*r);
kwadrat_opisany(x,y,round(r*sqrt(2)));
repeat until keypressed;
closegraph
end.
```

Spis treści

<u>JEZYKI PROGRAMOWANIA.....</u>	3
JEZYK PROGRAMOWANIA.	3
JEZYK MASZYNOWY.	3
ASSEMBLER	3
JEZYKI WYŻSZEGO POZIOMU.....	4
PROGRAMY INTERPRETUJĄCE (INTERPRETATORY).....	5
PROGRAMY KOMPILUJĄCE (KOMPILATORY).....	5
INTERPRETACJA A KOMPILACJA.....	6
<u>ELEMENTY JEZYKA PROGRAMOWANIA.....</u>	7
SŁOWA KLUCZOWE.	7
IDENTYFIKATOR.....	7
DEKLARACJA.....	7
OPERATORY.	7
PROCEDURY I FUNKCJE.	9
<u>ETAPY TWORZENIA PROGRAMU.</u>	10
PROBLEM DO ROZWIĄZANIA.....	10
ALGORYTM.....	10
REDAGOWANIE PROGRAMU	11
<u>PROGRAM I JEGO ELEMENTY.....</u>	12
TYPY, STAŁE I ZMIENNE	12
INSTRUKCJE.....	12
PROCEDURY, FUNKCJE I MODUŁY.....	13
<u>STRUKTURA PROSTEGO PROGRAMU I JEGO ANALIZA.</u>	16
<u>OBSŁUGA EDYTORA</u>	20
OPERACJE NA OKIENKACH.....	20
OPERACJE WYKONYWANE NA OKIENKACH	20
SPROWADZANIE PROGRAMU Z DYSKU DO EDYTORA.	21
KOMPILOWANIE PROGRAMU.	22
KŁAWISZE REDAKCYJNE.	22
<u>STRUKTURA PROGRAMU.....</u>	24
NAGŁÓWEK PROGRAMU	24
BŁOK.....	24
STRUKTURA PROGRAMU.....	24

DEKLARACJE MODUŁÓW.....	24
DEKLARACJE ETYKIET.....	25
DEFINICJE NAZW LITERAŁÓW.....	25
DEFINICJE TYPÓW.....	26
DEKLARACJE ZMIENNYCH.....	26
DEFINICJE ORAZ DEKLARACJE FUNKCJI I PROCEDUR.....	26
<u>TYPY DANYCH.....</u>	<u>28</u>
<u>INSTRUKCJE.....</u>	<u>33</u>
<u>STANDARDOWE WEJŚCIE – WYJŚCIE.....</u>	<u>34</u>
<u>PIERWSZE PROSTE PROGRAMY.....</u>	<u>35</u>
<u>PROSTE PROGRAMY Z WYKORZYSTANIEM ITERACJI.....</u>	<u>43</u>
<u>GRAFIKA-MODUŁ <i>GRAPH</i>.....</u>	<u>51</u>
PRZEDFINIOWANE IDENTYFIKATORY MODUŁU <i>GRAPH</i>	51
STAŁE OZNACZANIA KOLORÓW I LICZBY KOLORÓW.....	51
STAŁE OKREŚLENIA RODZAJU I GRUBOŚCI LINII.....	52
STAŁE DEFINIOWANIA ZNAKÓW WYPEŁNIAJĄCYCH KONTURY.....	52
STAŁE KROJÓW PISMA, KIERUNKU WYPROWADZANIA TEKSTÓW I ROZMIARU LITER.....	53
STAŁE JUSTOWANIA TEKSTU W POZIOMIE I PIONIE.....	53
STAŁE OGRANICZANIA GRAFIKI DO BIEŻĄCEGO OKNA.....	54
PROCEDURY INICJUJĄCE I ZAMYKAJĄCE TRYB GRAFICZNY.....	54
PROCEDURY USTALAJĄCE PARAMETRY GRAFICZNE.....	55
PROCEDURY GRAFICZNE.....	55
FUNKCJE I PROCEDURY TEKSTOWE.....	58
INNE.....	59
<u>PROGRAMY GRAFICZNE.....</u>	<u>60</u>
<u>INSTRUKCJE STRUKTURALNE.....</u>	<u>70</u>
<u>PROGRAMY Z WYKORZYSTANIEM INSTRUKCJI ITERACYJNYCH.....</u>	<u>77</u>
<u>PROGRAMY Z PROCEDURAMI I FUNKCJAMI.....</u>	<u>89</u>

*OPRACOWAŁA: MGR JUSTYNA JESIOTR
ZESPÓŁ SZKÓŁ NR 3 W WAŁCZU*