INFORMATYKA

Programowanie w języku Pascal

opracowanie : mgr inż. Dominik Sieciński
nauczyciel mianowany

Spis treści

1. Podstawowe definicje

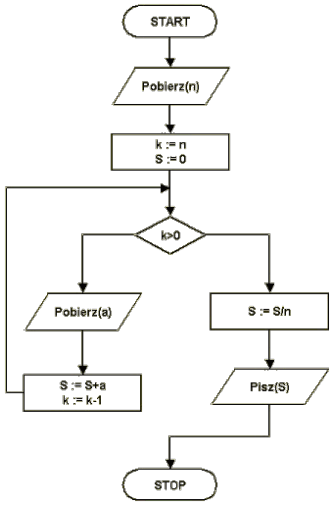
Algorytm - skończony i uporządkowany ciąg jasno zdefiniowanych czynności konieczny do wykonania pewnego zadania. Często jako przykład algorytmu podaje się przepis kulinarny, czy instrukcje obsługi/postępowania. Jednakże, aby taki przepis/ instrukcja był algorytmem muszą być spełnione ważne warunki:

- *skończony* - czyli musi istnieć możliwość przejścia od pierwszego do ostatniego punktu algorytmu. Nie może być sytuacji, że wykonując kolejne czynności "zapętlimy się", czyli będziemy wykonywać określone czynności bez końca. Przykładem takiego błędnego algorytmu jest zdanie: "*Nanieść szampon na włosy, myć, spłukać, czynność powtórzyć*". Poprawne zdanie powinno brzmieć: "*Nanieść szampon na włosy, myć, spłukać, w razie potrzeby czynność powtórzyć*".

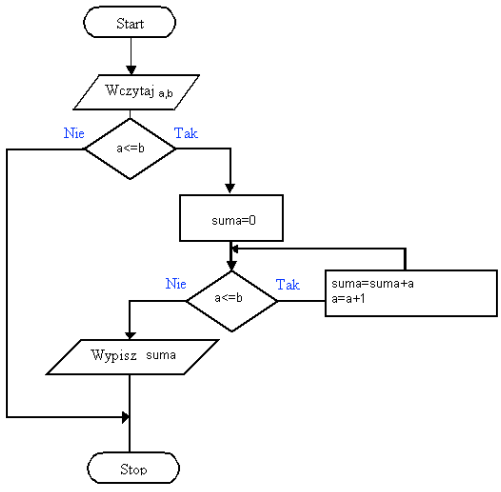
- *uporządkowany* - czyli ważna jest kolejność wykonywania kroków. Jeśli ją zmienimy, efekt końcowy będzie inny od oczekiwanego. Przykładem może być przepis kuchenny: "*rozgrzej patelnię, następnie mieszaj składniki przez 10 minut*". Zmiana kolejności spowoduje zmianę efektu końcowego: "*mieszkaj składniki przez 10 minut, następnie rozgrzej patelnię*".

- *jasno zdefiniowane czynności* - czyli, że opis danej czynności musi być dokładny. Prawidłowym zdaniem jest więc "*dolej szklankę mleka*", a błędnym "*dolej trochę mleka*".

Schemat blokowy - to zapis algorytmu w postaci schematycznego rysunku zamiast w postaci listy kroków. Oznaczenia poszczególnych bloków są umowne, ale przyjęło się, że prostokąt oznacza określone działanie, pochylony prostokąt wprowadzanie lub wyświetlanie danych, romb oznacza pytanie na które są dwie odpowiedzi - tak/nie, natomiast elipsa w zależności od napisu w środku oznacza START (początek) lub STOP (koniec) algorytmu.



błędny schemat (brak opisu TAK/NIE przy rombie)



poprawny schemat

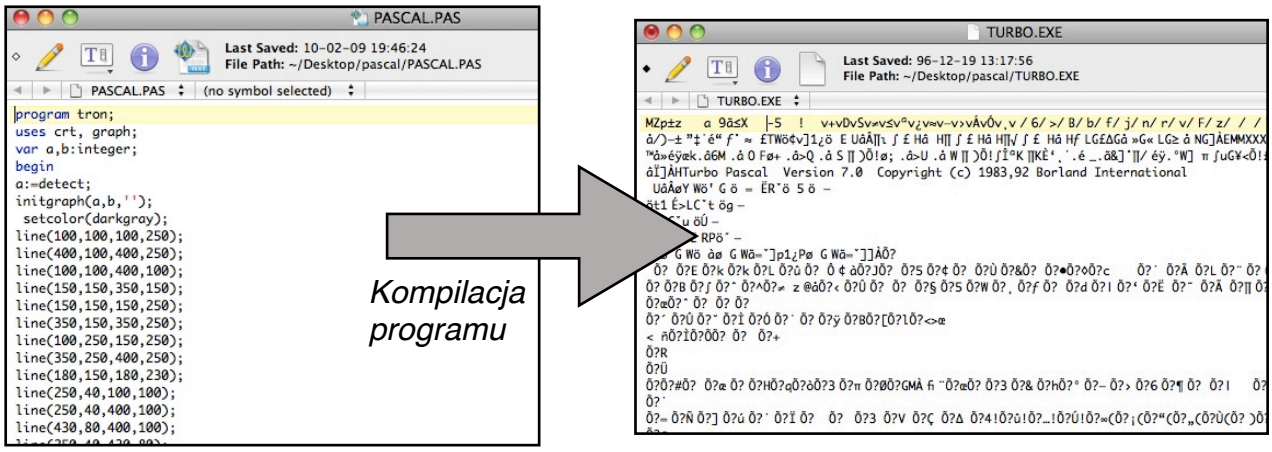
Kod źródłowy - to zapis algorytmu w wybranym języku programowania. Wybór języka programowania zależy wyłącznie od wiedzy i upodobań programisty oraz od tego co chcemy zrobić. Np. Clarion czy dBase są językami specjalizowanym w bazach danych, o Pascalu mówi się, że jest idealny do nauki programowania, PHP sprawdza się przy zastosowaniach internetowych, a np. RealBasic pozwala na pisanie programów przeznaczonych pod różne systemy operacyjne.

Przykłady kodu wypisującego napis "Hello, world!" na ekranie w różnych językach:

BASIC	PASCAL	PHP
10 PRINT "Hello, world!" 20 END	Begin WriteLn('Hello, world!'); End.	<?php echo "Hello, world!"; ?>

Kompilacja - proces polegający na zamianie kodu źródłowego na kod zrozumiały dla komputera (kod wynikowy, program). Po kompilacji program działa "niezależnie" od języka programowania, w którym był przygotowany. Kompilacja jest wymagana zawsze przed uruchomieniem programu. Kompilatory zwykle też wyłapują w momencie kompilacji błędy w zapisie kodu źródłowego. Jednak nie wszystkie języki programowania wymagają kompilacji. Niektóre z nich (np. starsze wersje Basica, język PHP) są interpretowane, czyli "kompilowane i wykonywane" na bieżąco, instrukcja po instrukcji. Wadą tego rozwiązania jest trudość ukrycia kodu źródłowego programu dla osób, które go zobaczyć nie powinny.

Kod źródłowy i jego postać po kompilacji



Kod źródłowy (Pascal)

Program - kod wynikowy (po kompilacji) możliwy do samodzielnego uruchomienia przez komputer pod wybranym systemem operacyjnym (w tym przypadku DOS)

Program - kod zrozumiały dla komputera, który powstaje z kodu źródłowego po procesie kompilacji (tzw. kod wynikowy). W przypadku systemów firmy Microsoft są to pliki z rozszerzeniem EXE, a przypadku systemów firmy Apple to pliki z rozszerzeniem APP. W programie nie widzimy kodu źródłowego

Zmienna - wydzielona część pamięci komputera, która posiada swoją nazwę, swój typ (np. czy jest to liczba, czy litera czy napis) oraz wartość (np. czy jest to liczba *12,5*, czy jest to literka *A*, czy napis *LICEUM*). Zmienne są potrzebne, aby napisać większość algorytmów. W matematyce odpowiednikiem zmiennych są “wiadome” i “niewiadome” w zadaniach (np. x,y,a itd).

W niektórych państwach, np. w USA istnieje prawna możliwość opatentowania algorytmu. Jednak dużo osób związanych z informatyką twiedzi, że spowalnia to rozwój tej dziedziny nauki i powoduje możliwość powstania monopolu na pewne rozwiązania informatyczne. Należy również wspomnieć, że istnieje techniczna możliwość dekompilacji programu spowrotem do kodu źródłowego (a przynajmniej przybliżonej jego postaci) i w ten sposób sprawdzenia jak wymyślony został algorytm, ale prawo zwykle zabrania takiego działania.

2. Zmienne i ich wartości

Chcąc zapisać algorytm w wybranym języku programowania potrzebne są zmienne. Pascal (w przeciwieństwie np. do PHP) wymaga zadeklarowania zmiennych jakich będziemy używać w algorytmie. Wymaga też podania ich typów. Podstawowe typy zmiennych jakie będą używane przy omawianiu algorytmów to:

byte - liczba całkowita dodatnia z zakresu od 0 do 255
integer - liczba całkowita dodatnia lub ujemna z zakresu -32.000 do 32.000
real - liczba rzeczywista (np. ułamek)
char - tylko jedna literka (lub inny znak dostępny na klawiaturze)
string - napis składający się maksymalnie z 255 znaków
typy tablicowe, rekordy i typy plikowe - omówione w innych rozdziałach

DEKLARACJA ZMIENNYCH

Przed zapisaniem każdego algorytmu należy zadeklarować zmienne jakie będą w tym algorytmie potrzebne. Deklaracja odbywa się poprzez podanie komendy *var*, np.:

```
var  a: integer;
     slowo: string;
     litera: char;
```

Taka deklaracja oznacza, że w algorytmie będziemy używać trzech zmiennych. W zmiennej *a* będziemy przechowywać liczbę całkowitą, w zmiennej *slowo* będziemy przechowywać dowolny napis, a w zmiennej *litera* jeden znak.

PROSTE OPERACJE NA ZMIENNYCH ZNAKOWYCH (char, string)

```
var  slowo: string;
     litera: char;
```

Najważniejszą operacją na zmiennych znakowych jest ich przypisywanie, dodawanie i zerowanie. Oto kilka przykładów:

slowo:='dowolny napis'; - umieszczenie w zmiennej *slowo* tekstu "dowolny napis"
litera:='a'; - umieszczenie w zmiennej litera znaku "a"

slowo:=''; - wyzerowanie zmiennej *slowo*, czyli umieszczenie pustego tekstu
litera:=''; - wyzerowanie zmiennej litera, czyli umieszczenie pustego znaku

Wartość zmienny znakowych zawsze podajemy w apostrofach. Komende kończymy zawsze znakiem średnika.

slowo:='ala'+' ma kota' - umieszczenie w zmiennej slowo napisu "ala ma kota"
litera:='x'; - umieszczenie w zmiennej *litera* znaku x
slowo:='ala'+litera; - umieszczenie w zmiennej *slowo* napisu "alax"

slowo:='ala'; - umieszczenie w zmiennej *slowo* napisu "ala"
slowo:=slowo+' ma kota'; - umieszczenie w zmiennej *slowo* napisu "ala ma kota"
w zmiennej *slowo* umieszczana jest wartość która tam już była plus tekst "ma kota"

Typowe błędy powodujące błąd przypisania (type mismatch):

litera:='x'+'ala ma kota'; - nie można umieścić w zmiennej *char* więcej niż 1 znaku
litera:='x'+'x'; - ten sam rodzaj błędu, zmienna char mieści max. 1 znak
litera:='xx' - 'x'; - operacja odejmowania nie jest możliwa na zmiennych znakowych

PROSTE OPERACJE NA ZMIENNYCH LICZBOWYCH (byte, real, integer)

Na zmiennych liczbowych możemy wykonywać większość operacji matematycznych, czyli dodawać, odejmować, mnożyć, dzielić, obliczać resztę z dzielenia, zaokrągląć itd. Oczywiście "*diabeł tkwi w szczegółach*" i ilość możliwych do popełnienia błędów jest znacznie większa niż w przypadku zmiennych znakowych.

```
var  b: integer;
     c: real;
     d: byte;
```

Przypisując wartość liczbową do zmiennej, nie używamy apostrofów, jak w przypadku zmiennych znakowych.

b:=5; - umieszcza wartość 5 w zmiennej *b*
c:=3.5; - umieszcza wartość 3 i pół w zmiennej *c* (zamiast przecinka podajemy kropkę)
d:=200; - umieszcza wartość 200 w zmiennej *d*

Typowe błędy powodujące błąd przypisania (type mismatch):

b:=2.5; - przypisanie ułamka do zmiennej typu integer (liczby całkowitej)
d:=400; - przypisanie wartości zmiennej spoza dozwolonego zakresu
b:=40000; - przypisanie wartości zmiennej spoza dozwolonego zakresu
b:=c; - przepisanie wartości zmiennej c (rzeczywistej) do b (do całkowitej)
b:=4/2; - mimo, że wynik jest całkowity, to kompilator uzna, że to błąd, bo wynikiem jest liczba rzeczywista o wartości 2.0 (dwa przecinek zero);

3. Zmiana wartości zmiennych

W trakcie działania algorytmu zmienne zwykle zmieniają swoje wartości. Aby dobrze zrozumieć na czym polega wykonywanie krok po kroku algorytmu należy przede wszystkim zrozumieć czym jest aktualna, poprzednia i następna wartość zmiennej. W tym celu posłużymy się prostym przykładem algorytmu, w którym będą występować trzy zmienne typu integer - a,b,c;

```
krok 0    var a,b,c: integer;
          begin {początek algorytmu zapisanego w Pascalu}
krok 1     a:=5;
krok 2     b:=1;
krok 3     c:=6;
krok 4     a:=b+a;
krok 5     c:=a-b;
krok 6     b:=4;
krok 7     a:=a+1;
          end.  {koniec algorytmu zapisanego w Pascalu}
```

Algorytm z definicji wykonywany jest krok po kroku z zachowaniem kolejności. W powyższym przykładzie mamy trzy zmienne - a,b,c oraz 7 kroków. W każdym kroku wartość jednej ze zmienych jest zmieniana.

krok 0	*a=?*	*b=?*	*c=?*	*(deklarujemy zmienne - nie mają jeszcze wartości)*
krok 1	*a=5*	*b=?*	*c=?*	*(zmienna a ma już wartość, pozostałe nie)*
krok 2	*a=5*	*b=1*	*c=?*	*(**WAŻNE**: zmienna a ma nadal wartość 5)*
krok 3	*a=5*	*b=1*	*c=6*	*(zmienne a i b mają wartość z kroku 1 i 2)*
krok 4	*a=6*	*b=1*	*c=6*	*(zmienna a w kroku 4 to aktualna wartość a w kroku poprzednim, czyli 5 plus aktualna wartość b w kroku poprzednim - 1. Ich suma jest wpisana od teraz jako wartość zmiennej a w kroku 4. Wartości zmiennych b i c nie ulegają zmianie)*
krok 5	*a=6*	*b=1*	*c=4*	*(wartość zmiennej c od teraz to wynik działania a-b. Wartość jaką miało c w poprzednim kroku nie ma w tym momencie znaczenia)*
krok 6	*a=6*	*b=4*	*c=4*	*(w zmiennej b jest umieszczona wartość 4, a poprzednia wartość b nie ma w tym momencie znaczenia)*
krok 7	*a=7*	*b=4*	*c=4*	*(nowa wartość a to a z poprzedniego kroku plus 1)*

Wartość zmiennych po wykonaniu całego algorytmu to *a=7, b=4, c=4*. Należy zwrócić uwagę, że w trakcie działania algorytmu wartości zmiennych ulegają zmianie.
Typowym błędem popełnianym w tego typu algorytmach byłoby umieszczenie np. w kroku nr 2 instrukcji *b=a+c*. Błąd polegałby na tym, że wiemy co prawda ile jest *a*, ale nie wiemy jeszcze jaką wartość ma zmienna *c*, więc nie można policzyć ile będzie wynosiło *b*.

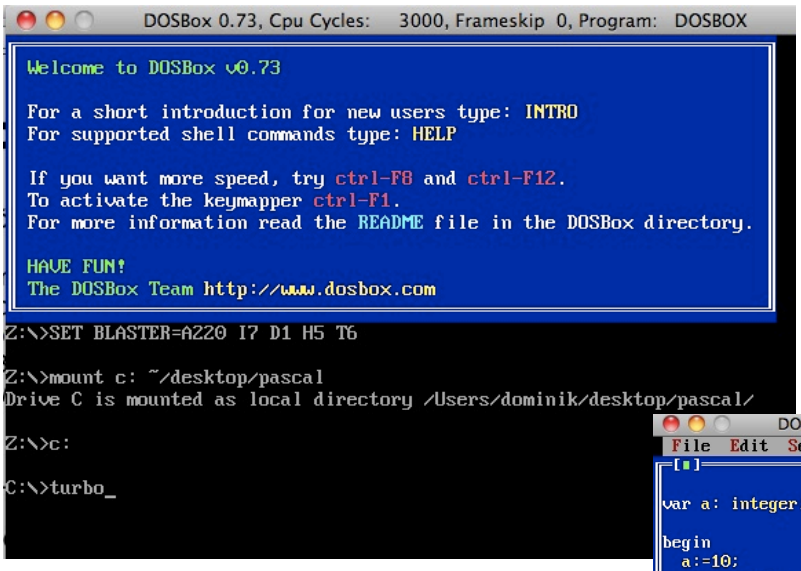
4. Pascal dla Apple

Aby skompilować i uruchomić na komputerze napisany przez nas program musimy najpierw zaintalować i uruchomić na komputerze edytor i kompilator Pascala. Istnieje wiele wersji Pascala, od darmowego kompilatora Free Pascal po Turbo Pascala firmy Borland. O ile sama instalacja pakietu nie stanowi większego problemu pod Windows o tyle na komputerach **Apple** (rówież w niektórych przypadkach w **Windows Vista**) należy użyć oprogramowania DOSBOX "udającego" środowisko MS-DOS dla którego był stworzony Pascal. DOSBOX zapewni pełną zgodność zwłaszcza przy programowaniu grafiki.

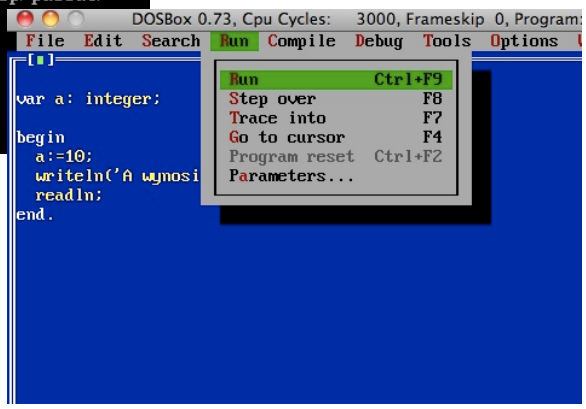
URUCHAMIANIE BORLAND PASCAL na komputerach APPLE:

Zakładając, że na pulpicie mamy folder PASCAL w którym znajduje się plik TURBO.EXE (czyli edytor i kompilator Pascala) procedura uruchomienia środowiska programistycznego jest następująca:

1. Uruchamiamy DOSBOX
2. Wpisujemy komendę: **mount c: ~/desktop/pascal** i naciskamy ENTER
3. Wpisujemy komendę: **c:** i naciskamy ENTER
4. Wpisujemy komendę: **turbo** i naciskamy ENTER



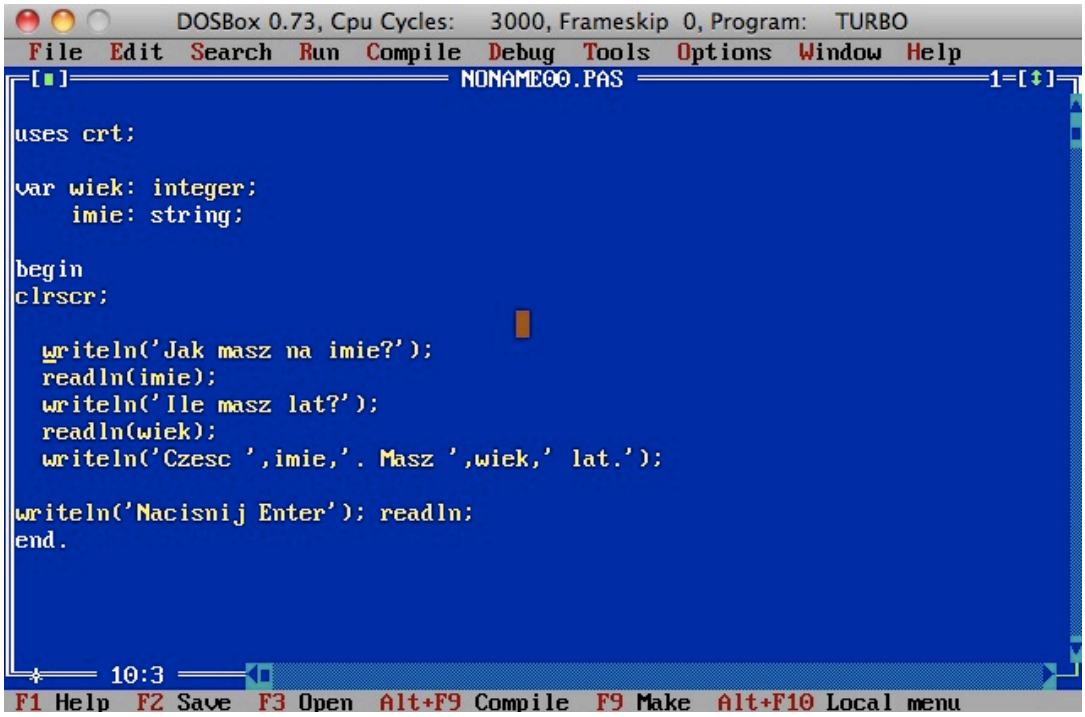
uruchomiony DOSBOX



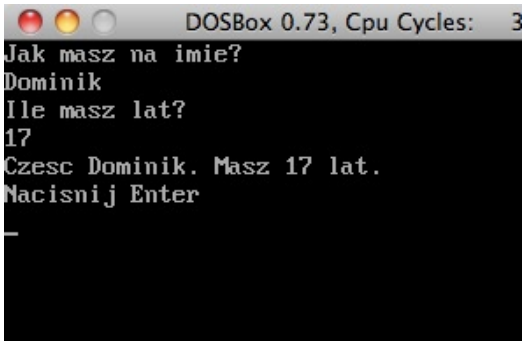
TURBO PASCAL gotowy do pracy

5. Mój pierwszy program

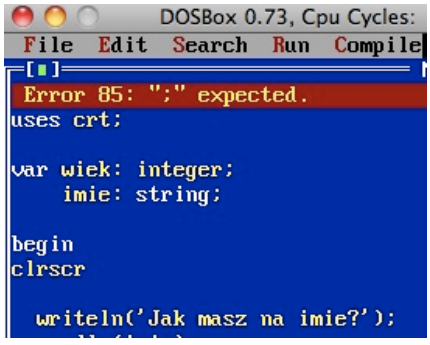
Po uruchomieniu Pascala możemy napisać swój pierwszy algorytm. Program spyta się o imię i wiek a następnie wyświetli na ekranie pozdrowienia.



Po bezbłędnym przepisaniu programu można spróbować go uruchomić wybierając w menu opcję RUN. Jeśli program zawiera błędy w zapisie, kompilator poinformuje o tym napisem na czerwonym pasku podając numer błędu oraz ustawiając kursor w miejscu, w którym prawdopodobnie jest błąd. Jeśli program błędów w zapisie nie zawiera zostanie skompilowany i uruchomiony.



poprawnie uruchomiony program



popularny błąd - brak średnika

6. Spis błędów przy kompilacji

Pisząc program w Pascalu można popełnić dwa rodzaje błędów. Błędy w poprawnym zapisie oraz błędy w algorytmie. Pierwszy rodzaj błędów wyłapuje kompilator w momencie próby uruchomienia programu podając ich numer na czerwonym pasku. Poniżej znajduje się spis najważnieszych komunikatów błędów wraz z ich tłumaczeniem.

1. Out of memory - Brak pamięci **2. Identifier expected** - Oczekiwano identyfikatora **3. Unkown identifier** - Nieznany identyfikator **4. Duplicate identifie** - Podwójny identyfikator **5. Syntax error -** Błąd składni **8. String constant exceeds line** - Stała łańcuchowa przekracza wiersz **14. Invauld file name** - Błędna nazwa pliku **15. File not found** - Nie znaleziono pliku **16. Disk full** - Dysk jest pełny **20. Variable identifier expected** - Oczekiwano identyfikatora zmiennej **21. Error in type** - Błąd w deklaracji typu **26. Type mismatch** - Niezgodność typów **29. Ordinal type expected** - Oczekiwano typu porzadkowego **30. Integer constant expected** - Oczekiwano stałej całkowitej **31. Constant expected** - Oczekiwano stałej **32. Integer or real constant expected** - Oczekiwano stałej calkowitej lub rzeczywistej **33. Pointer Type identifier expected** - Oczekiwano identyfikatora typu wskaźnikowego **34. Invalid function result type** - Błędny typ wyniku funkcji **35. Label identufier expected** - Oczekiwano identyfikatora etykiety **36. BEGIN expected** - Oczekiwano BEGIN **37. END expected** - Oczekiwano END **38. Integer expression expected** - Oczekiwano wyrażenia całkowitego **39. Ordinal expression expected** - Oczekiwano wyrażenia porządkowego **40. Boolean expression expected** - Oczekiwano wyrażenia logicznego **42. Error in expression** - Błąd w wyrażeniu **50. DO expected** - Oczekiwano DO **54. OF expected** - Oczekiwano OF **57. THEN expected** - Oczekiwano THEN **58. TO or DOWNTO expected** - Oczekiwano TO lub DOWNTO **62. Devision by zero** - Dzielenie przez zero **64. Cannt read or write variables of this type** - Błędny typ argumentu **66. String variable expected** - Oczekiwano zmiennej łańcuchowej **67. String expression expected** - Oczekiwano wyrażenia łańcuchowego **76. Constant out of range** - Stała poza zakresem **85. ; expected** - Oczekiwano średnika **86. : expected** - oczekiwano dwukropka **87. , expected** - Oczekiwano przecinka **88. (expected** - Oczekiwano nawiasu początkowego **89.) expected** - Oczekiwano nawiasu kończącego **90. = expected** - Oczekiwano znaku różności **91. := expected** - Oczekiwano znaku przypisania **94. . expected** - Oczekiwano kropki **95. .. expected** - Oczekiwano dwóch kropek **97. Invalid FOR control variable** - Błędna zmienna sterująca w instrukcji FOR **106. Character expression expected** - Oczekiwano wyrazenia znakowego **112. CASE constant out of range** - Stała wybory poza zakresem **113. Error in statement** - Błąd w instrukcji **132. Critical disk error** - Poważny błąd na dysku **143. Invalid procedure or function reference** - Błędne odwołanie do procedury.

7. Wczytywanie i wyświetlanie

Aby napisać uniwersalny program konieczna jest możliwość podania wartości zmiennych przez użytkownika programu oraz wyświetlenie na ekranie wyników działania algorytmu. W Pascalu służą do tego trzy procedury: *readln* (wczytywanie) oraz *writeln* i *write* (wyświetlanie).

Przykład:

```
var wiek: integer;          - deklaracja zmiennej
begin                       - początek algorytmu
  writeln('Ile masz lat?');  - wyświetla na ekranie napis "Ile masz lat?"
  readln(wiek);              - czeka na wprowadzenie wartości zmiennej
                               wiek i naciśnięcie klawisza Enter. Po wprowadzeniu
                               wartości dopiero od tego momentu w algorytmie
                               komputer zna wartość zmiennej wiek
  readln;                    - czeka na wprowadzenie samego klawisza Enter
end.                         - koniec algorytmu
```

Po uruchomieniu programu edytor i kompilator Pascala przełączy widok na tzw. USER SCREEN czyli na drugi ekran na którym pokazuje wynik działania algorytmu. Po zakończeniu wykonywania programu automatycznie wraca do edycji (niebieski ekran). Aby zdążyć zobaczyć wynik działania algorytmu przez instrukcją *end.* należy wprowadzić instrukcję readln; która nie wczytuje żadnych danych do pamięci komputera, a czeka wyłącznie na naciśnięcie klawisza Enter.

Kilkukrotne uruchomienie programu powoduje, że na USER SCREEN pojawia się tak wiele napisów, że po chwili nie wiadomo już, który napis dotyczył którego uruchomienia programu. Dlatego dobrze jest dodać instrukcję czyszczenia ekranu przed rozpoczęciem algorytmu. Robi się to w następujący sposób:

```
uses crt;                   - wczytanie modułu (zestawu dodatkowych
                              instrukcji), który wytłumaczy Pascalowi, na czym
                              polega czyszczenie ekranu
var wiek: integer;          - deklaracja zmiennej
begin                       - początek algorytmu
clrscr;                     - instrukcja wyczyszczenia ekranu USER SCREEN
                              (skrót od angielskich słów clear screen)
  writeln('Ile masz lat?');  - wyświetla na ekranie napis "Ile masz lat?"
  readln(wiek);              - czeka na wprowadzenie wartości zmiennej
                               wiek i naciśnięcie klawisza Enter. Po wprowadzeniu
                               wartości dopiero od tego momentu w algorytmie
                               komputer zna wartość zmiennej wiek
  readln;                    - czeka na wprowadzenie samego klawisza Enter
end.                         - koniec algorytmu
```

PRZYKŁADY UŻYCIA INSTRUKCJI WCZYTANIA i WYPISANIA WARTOŚCI ZMIENNYCH

```
var a: integer;
    b: real;
    c: string;
    d: char;
```

readln(a); - wczytanie wartości zmiennej a

Wyświetlanie zmiennej typu real:

writeln(b); - wyświetlenie wartości zmiennej typu real. Wartość zmiennej real wyświetlana jest zawsze w **notacji naukowej**, która jest mało czytelna dla użytkownika, dlatego zwykle przy wyświetlaniu zmiennej typu real wymusza się na Pascalu wyświetlanie wartości zmiennej w zwykłej notacji: **writeln(b:0:10);**

b:=10.234
writeln(b); - spowoduje wyświetlenie liczby 1.0234000000E+01
czyli 1.0234000000 razy 10 do potęgi 1

writeln(b:0:2); - spowoduje wyświetlenie liczby 10.23 (dwa miejsca po przecinku)
writeln(b:0:4); - spowoduje wyświetlenie liczby 10.2340 (cztery po przecinku)

b:**0**:4 - zero oznacza, że przed przecinkiem zostanie przydzielone tyle miejsca na ekranie, aby zmieściła się cała wartość liczby. Zamiast zero można podać inną liczbę, wtedy wyświetlanie zostanie wyrównane do prawej.

Wyświetlanie zmiennych typu integer z wyrównaniem do prawej

Podobnie jak w zmiennych real można użyc wyrównania pozycji liczby przy zmiennych typu integer. Wyjaśnia to przykład:

Zapis algorytmu	**Efekt wyświetlania na ekranie:**
a:=200;	
b:=2000;	
d:=2;	
writeln(a);	200
writeln(b);	2000
writeln(d);	2

Skoro najwięsza liczba z wyświetlonych ma 4 znaki, możemy liczby typu integer wyrównać do prawej, używając podobnej metody jak przy wyświetlaniu liczby real. Ten sposób wyświetlania będzie bardzo pomocny przy zmiennych typu tablicowego.

Zapis algorytmu **Efekt wyświetlania na ekranie:**

```
a:=200;
b:=2000;
d:=2;
writeln(a:4);
writeln(b:4);
writeln(d:4);
```

```
 200
2000
   2
```

Różnica między write i writeln (ln = new line).

Użycie instrukcji write powoduje wyświetlenie wartości zmiennej. Po wyświetleniu wartości kursor zostaje w miejscu za wyświetloną wartością. A więc każde następne wyświetlenie wartości dowolnej zmiennej pojawi się w tej samej linijce.

Przykład 1: **Efekt działania na ekranie:**

```
a:=5; b:=10.2345;
write(a);
writeln(b:0:4);
write('ala ma kota');
writeln(' kot ma ale');
```

```
510.2345 (brak Entera po wypisaniu a)
ala ma kota kot ma ale
```

Przykład 2: **Efekt działania na ekranie:**

```
a:=5; b:=10.2345;
writeln(a);
writeln(b:0:4);
writeln('ala ma kota');
writeln(' kot ma ale');
```

```
5
10.2345
ala ma kota
 kot ma ale (spacja w napisie na początku)
```

Wypisanie napisów wraz z wartościami zmiennych

Aby na ekranie pojawił się napis zawsze należy użyć znaków apostrofu:
writeln('a'); - pojawi się napis "a" ponieważ użyto apostrofów

Aby wypisać wartość zmiennej nie używamy apostrofów
writeln(a); - pojawi się liczba 5, ponieważ zmienna a ma wartość 5

Instrukcji write/writeln można użyć też w wersji złożonej:

writeln('ala ma ',a,' lat i ma ',a,' kotów'); - wyświetli napis ala ma 5 lat i ma 5 kotów a za napisem zrobi enter, czyli następny napis który pojawi się na ekranie będzie w nowej linijce.

write**ln**(**'ala ma '**,a,**' lat i ma '**,a,**' kotów'**); - napisy są w apostrofach, zmienne bez oddzielone przecinkami od napisów

Dodatek - Pliki tekstowe (podstawy)

Program napisany w Pascalu przechowuje wartości zmiennych w pamięci RAM. Wadą tego rozwiązania jest fakt, że po zakończeniu programu, lub po np. przypadkowym wyłączeniu prądu, wprowadzone dane znikają bezpowrotnie.

Większość programów potrafi przechowywać swoje dane na dysku twardym komputera. Taką możliwość mają również programy napisane w języku Pascal. Służą do tego zmienne plikowe, w tym ich najprostrza wersja - pliki tekstowe.

Aby umożliwić zapisywanie wartości zmiennych (w tym przypadku zmiennej typu string) należy zadeklarować odpowiednio zmienne.

```
var   a: text;              - zmienna typu plik tekstowy
      napis: string;        - zmienna w której możemy przechowywać napis do 255 znaków
```

Następnie w programie należy przypisać odpowiednio zmienną do pliku i utworzyć na dysku plik, w którym będą trzymane nasze dane (napisy).

```
begin
      assign(a,'plik.txt');          - przypisanie nazwy pliku do zmiennej "a"
      rewrite(a);                    - fizyczne utworzenie pliku plik.txt na dysku
                                       plik zostanie utworzony w tym samym folderze
                                       jeśli taki plik już istnieje - zostanie on skasowany
                                       w którym znajduje się plik .PAS. Plik będzie miał
                                       wielkość zero bajtów.
            napis:='ala ma kota';    - wpisanie do zmiennej napis wartości "ala ma kota"
            writeln(napis);          - wypisanie na ekranie napisu "ala ma kota"
            writeln(a,napis);        - wrzucenie do pliku plik.txt wartości ze zmiennej napis
                                       oraz znaku ENTER (nowa linia).
      close(a);                      - zamknięcie pliku (zwykle w tym momencie wszystkie
                                       wysłane do pliku napisy są dopiero fizycznie zapisywane)
end.
```

Po wykonaniu powyższego programu na dysku pojawi się plik, który po otwarciu w programie notatnika (lub np. TextWrangler - w przypadku komputerów Apple) będzie zawierał napis "ala ma kota".

WAŻNE UWAGI:

Użycie instrukcji **write(a,napis)** nie jest zalecane z uwagi na brak znaku ENTER na końcu linijki. Po otwarciu pliku w notatniku czytelność danych jest znikoma.

Na komputerach Apple należy również zwrócić uwagę, że znaki ENTER w większość programów zapisywane są inaczej niż w systemie Windows. W programie TextWrangler należy, aby uniknąć błędów odczytu pliku, wybrać opcję Windows przed zapisem plików (patrz obrazek powyżej).

Istnieje możliwośc odczytania danych z pliku, podobnie jak odczytujemy wartość zmiennych z klawiatury - instrukcją readln. Nie możemy jednak użyć wtedy instrukcji *rewrite(a)*, która powoduje zniszczenie już istniejącego pliku. Należy użyć wtedy instrukcji *reset(a).*

```
begin
    assign(a,'plik.txt');        - przypisanie nazwy pliku do zmiennej "a"
    reset(a);                    - otwarcie pliku plik.txt do odczytu
                                   uwaga plik musi już istnieć na dysku
                                   i zawierać przynajmniej jedną linijke tekstu wraz
                                   ze znakiem enter
        readln(a,napis);         - odczytanie z pliku pierwszej linijki wraz ze znakiem enter
        writeln(napis);          - wypisanie na ekranie odczytanej z pliku linijki tekstu.
    close(a);                    - zamknięcie pliku
end.
```

Różnica w przedstawionych programach dotyczy przede wszystkim instrukcji powodującej otwarcie pliku. *Rewrite(a)* powoduje zawsze zniszczenie istniejącego pliku (jeśli istnieje) i założenie nowego. *Reset(a)* powoduje otwarcie pliku, który już istnieje, bez niszczenia jego zawartości (musi istnieć, bo inaczej program zakończy się błędem). Otwarcie pliku komendą Reset(a) pozwala nie tylko **na odczytanie jego zawartości (readln(a,napis)), ale również na zapis nowych danych (writeln(a,napis))** do pliku.

Sytuacja jest o tyle nie wygodna, że jeśli plik nie istnieje, to musimy użyć instrukcji rewrite. A jeśli plik z danymi (np. wprowadzonymi dzień wcześniej) już jest, nie możemy użyć instrukcji rewrite, gdyż skasuje to nasze wprowadzone wcześniej dane. Istnieje jednak rozwiązanie tego problemu. Dzięki derektywom kompilatora (zapisywanych za pomocą klamr i znaku dolara) istnieje możliwość napisania programu który zadziała w następujący sposób:

1. przypisz nazwę pliku do zmiennej a
2. spróbuj otworzyć plik (**reset**)
3. jeśli nie udało się otworzyć pliku oznacza to, że on nie istnieje
4. jeśli plik nie istnieje to utworz pusty plik (**rewrite**)

```
var a: text;
    napis: string;

begin
    assign(a,'plik.txt');        - przypisanie nazwy pliku do zmiennej "a"
    {$I-}                        - wyłączenie obsługi błędów powodujących
                                   zatrzymanie programu
    reset(a);                    - próba otwarcia istniejącego pliku plik.txt
    if IOResult <>0 then rewrite(a);   - sprawdzenie czy wystąpił błąd przy otwarciu
                                   pliku plik.txt - jeśli wystąpił to oznacza, że pliku
                                   nie było na
dysku. Wtedy wykonanie instrukcji
                                   rewrite(a) -


która taki plik utworzy
    {$I+}                        - p o n o w n e
włączenie obsługi błędów
    …….                          - zapis lub odczyt
```

danych z pliku (reszta programu)
close(a); - zamknięciu dostępu do pliku przez program
end.

Dopisanie danych na końcu pliku

Jeśli istnieje już plik na dysku i użyjemy instrukcji reset(a) plik zostanie otwarty. Możliwe jest odczytanie danych znajdujących się w nim (readln(a,napis)) oraz zapisanie danych do pliku (writeln(a,napis)). Niestety po otwarciu pliku tzw. znacznik miejsca w którym jesteśmy w pliku ustawiany jest przed pierwszą linijką. Oznacza to, że próba odczytu danych z pliku spowoduje odczytanie pierwszej linijki. Ponowna próba odczytu spowoduje odczytanie drugiej linijki itd. Natomiast próba zapisu do pliku po jego otwarciu instrukcją reset(a) spowoduje nadpisanie nowych danych na istniejącą już linijke. Aby tego uniknąć plik należy otwierać instrukcją append(a) zamiast reset(a).

```
begin
    assign(a,'plik.txt');          - przypisanie nazwy pliku do zmiennej "a"
    append(a);                     - otwarcie istniejącego pliku na dysku (plik.txt)
                                     oraz ustawienie znacznika miejsca za ostatnią
                                     liniką w istniejącym pliku
        readln(napis)              - program pyta użytkownika o wartość zmiennej
                                     napis (np. "jak masz na imię?")
        writeln(napis);            - wypisanie na ekranie wartości zmiennej napis
        writeln(a,napis);          - wrzucenie do pliku plik.txt wartości ze zmiennej napis
                                     oraz znaku ENTER (nowa linia) za ostatnią
                                     istniejącą linijką
    close(a);                      - zamknięcie pliku (zwykle w tym momencie wszystkie
                                     wysłane do pliku napisy są dopiero fizycznie zapisywane)
end.
```

Z instrukcji **append** możemy skorzystać jedynie przy zmiennych typu text (pliki tekstowe). W przypadku innych typów plików wymagana będzie instrukcja **seek**.

Wyświetlenie całej zawartości pliku na ekranie (EOF)

Instrukcja EOF (End of File) pozwala na sprawdzenie, czy istnieje w pliku następna linijka. Dzieki niej możliwe jest pobranie całej zawartości pliku kolejno linijka po linijce. Oto rozbudowany przykład:

```
begin
    assign(a,'mojplik.txt');
    {$I-}                          - sprawdzamy czy istnieje plik, jeśli nie
    reset(a);                        informujemy o tym użytkownika
    if IOResult <>0 then begin
                        writeln('Plik mojplik.txt nie istnieje'); halt;
                    end;
    {$I+}
    repeat
        readln(a,napis);           - czyta linijkę z pliku wraz ze znakiem ENTER
        writeln(napis);            - wyświetla na ekranie przeczytaną linijkę
```

```
    until EOF(a);                    - sprawdza, czy w pliku jeszcze coś jest
                                       a jeśli tak, wraca do instrukcji repeat
readln;
end.
```

więcej na zajęciach z programowania :)))
oraz w materiałach na www.edukontakt.pl