Kurs Turbo Pascal 7.0

By Kajoj

PODSTAWY

1. Co to jest Pascal?

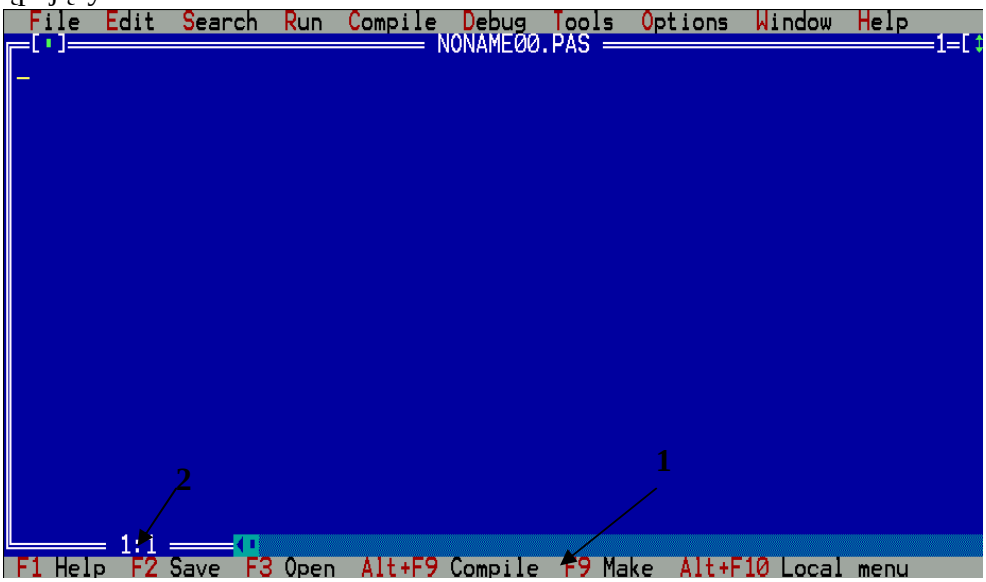
Pascal jest to język programowania stworzony przez Niklausa Wirtha (naukowiec z Zurychu) na podstawie języka ALGOL na przełomie lat ’60 i ’70. Niklausa Wirtha przedstawił Międzynarodowej Federacji Przetwarzania Informacji projekt nowego języka. Niestety projekt został odrzucony. Mimo to Pascal zyskał olbrzymią popularność, dzięki oferowaniu dużych możliwości obliczeniowych przy małych wymaganiach sprzętowych.

Pascal jest językiem bardzo uniwersalnym. Równie dobrze radzi sobie z przetwarzaniem dużych ilości danych, jak i ze skomplikowanymi symulacjami matematycznymi.

2. Uruchamiamy Turbo Pascala

Istnieje wiele wersji kompilatorów TP. My skupimy się na najpopularniejszym opracowanym przez firmę Borland i w wersji 7.0.

Wywołanie pliku bp.exe powoduje start programu. Po starcie na ekranie pojawia się następujący obraz:



Oto opis poszczególnych elementów ekranu:

1. **Dolna linia menu** zawiera opcja, która wybieramy:
   - myszą przez ustawieni kursora na nazwie i naciśnięcie LPM (Lewego Przycisku Myszy),
   - za pomocą klawiszy funkcyjnych lub kombinacji klawiszy ALT lub CTRL z innymi klawiszami (ang. hot keys).
2. **Współrzędne kursora tekstowego**
3. **Znak zamknięcia okna.**
4. **Górna linia menu** – oprócz opcji zawiera także podopcje, które wybieramy, naciskając kombinacje klawisza ALT z klawiszami pierwszych liter tych poleceń (F, E, S, R ...) lub klikając myszką.
5. **Domyślna nazwa pliku** – do niego zostanie zapisana treść stworzonego przez nas programu, ale tylko wtedy, gdy nie zechcemy nadać temu plikowi własnej nazwy.
6. **Znak zwiększania i zmniejszania okna**
7. **Okno edytora programów**
8. **Paski przesuwu**

Program wyłączamy poprzez naciśnięcie kombinacji ALT+X

3. Struktura programu

Program w języku Pascal składa się z następujących elementów:

```
program nazwa_programu;        {tytuł programu     }
begin                          {początek programu  }
     polecenie1;               {  \                }
     polecenie2;               {   treść programu  }
     polecenie3;               {  /                }
end.                           {koniec programu    }
```

Dyskusja:

Program w Pascalu rozpoczynamy tytułem, który deklarujemy po słowie *Program*. Każde polecenie w języku Pascal należy zakończyć średnikiem *;* Zasadnicza część programu zamyka się w słowach: *begin* i *end*. To właśnie polecenia z tej części będzie wykonywał program. Zwróć uwagę na wcięcia które ułatwiają czytanie kodu programu. Będzie to niezwykle przydatne kiedy twoje programy będą bardziej rozbudowane.

4. Moduły w języku Pascal.

Wszystkie polecenia języka Turbo Pascal zostały uporządkowane w postaci grup zwanych modułami (ang. units). Istnieje kilka standardowych modułów, które zostały dołączone do programu Pascal. Oprócz tego można tworzyć własne moduły, ale o nich później. Wśród modułów standardowych wyróżniamy:

- **System** – zawiera procedury standardowe języka, do których dostęp nie wymaga specjalnych zabiegów.
- **Crt** – zawiera procedury obsługi ekranu, klawiatury i sterowania dźwiękami emitowanymi przez głośnik komputera.
- **Dos –** zawiera procedury współpracujące z systemem operacyjnym.
- **Graph** – zawiera procedury potrzebne do tworzenia grafiki.

Pozostałe moduły standardowe noszą nazwy: Turbo3, Graph3, Printer i Overlay.

Aby móc korzystać z procedur któregoś z modułów należy zadeklarować jego użycie (nie dotyczy to modułu system, który jest automatycznie deklarowany przez program Pascal). Moduł deklaruje się poprzez użycie słowa *uses*. Poczym należy podać nazwę modułu lub modułów oddzielając je przecinkami.

```
program nazwa_programu;
uses crt,graph;
begin
     polecenie1;
     polecenie2;
     polecenie3;
end.
```

5. *Moduł Crt.*

Moduł Crt należy do najczęściej stosowanych modułów w TP, dlatego omówimy wszystkie jego procedury i funkcje.

Tabela 1

Nazwa	Proc/ Func*	Opis	Trudność**
AssignCrt(plik : text)	Proc	Łączy plik tekstowy z oknem (ekranem) CRT.	***
ClrEol	Proc	Wymazuje wszystkie znaki od miejsca w którym znajduje się kursor do końca linii.	*
ClrScr	Proc	Czyści ekran i ustawia kursor w lewym górnym rogu (pozycja 0:0).	*
Delay(MS : word)	Proc	Zatrzymuje program na określoną ilość milisekund.	*
DelLine	Proc	Usuwa linię, w której znajduje się kursor.	*
GotoXY(X, Y : byte)	Proc	Przesuwa kursor do punktu x:y.	*
HighVideo	Proc	Włącza wysoką intensywność barw i znaków.	*
InsLine	Proc	Wstawia pustą linię w miejscu gdzie znajduje się kursor.	*
KeyPressed { : Boolean }	Func	Sprawdza aktywność klawiatury.	**
LowVideo	Proc	Włącza niską intensywność barw i znaków.	*
NormVideo	Proc	Włącza normalną intensywność barw i znaków.	*
NoSound	Proc	Wyłącza komputerowego spikera.	*
ReadKey { : Char }	Func	Wczytuje znak z klawiatury.	*
Sound(Hz : word)	Proc	Włącza komputerowego spikera (dźwięk o częstotliwości podanej w hercach.	*
TextBackground(color : byte)	Proc	Zmienia kolor tła.	**
TextColor(color : byte)	Proc	Zmienia kolor tekstu.	**
TextMode(mode : integer)	Proc	Zmienia wygląd tekstu.	**
WhereX { : byte }	Func	Wczytuje poziomą (X) pozycje kursora.	**
WhereY { : byte }	Func	Wczytuje pionową (Y) pozycję kursora.	**
Window(X1, Y1, X2, Y2 : byte)	Proc	Tworzy nowe tekstowe okno.	***

*Procedura lub Funkcja.
**Poziom trudności w skali 1-5

Nadszedł czas na napisanie naszego pierwszego programu. Będzie on ilustrował sposób deklarowania użycia modułu Crt i zastosowanie niektórych opisanych procedur.

```
program 001;   {nazwa programu}
uses crt;      {deklaracja użycia modułu Crt}
begin
     TextBackground(7);  {zmienia kolor tła na szary}
     ClrScr; {czyścimy ekran}
     GotoXY(30,13); {umieszczamy kursor w pozycji 30:13}
```

```
        TextColor(4);  {zmieniamy kolor czcionki na czerwony}
        write('To jest moj pierwszy program'); {i piszemy}
end.
```

Dyskusja:

Jak zauważyliście kolory podajemy w postaci liczb (lub angielskich nazw kolorów – patrz tabela 2). Następny punkt na którym się zatrzymamy to polecenie *GotoXY*. W nawiasie podajemy współrzędne ekranu w jakich ma się znaleźć kursor (standardowe okno ma wymiary 80x25. Na koniec na pewno zwróciliście uwagę na polecenie *write*. Nie ma go w Tabeli 1. Dlaczego? Ponieważ nie należy ono do modułu Crt tylko do modułu System, którego, jak już mówiłem, nie deklaruje się na początku programu. Procedura *write* ma dosyć skomplikowaną budowę:

a) dla typów plików:
   Write(F, V1[, V2, ... , Vn])
b) dla plików tekstowych:
   Write([var F : text;] P1 [, P2, ... , Pn])

ale w gruncie rzeczy zapisuje znaki do pliku i jest to polecenie bardzo proste w użyci. W tym przypadku nie podaliśmy żadnego pliku więc program napisał nasz ciąg znaków na ekranie monitora. Pamiętaj tylko, że tekst jaki ma być napisany na ekranie musi być zawarty między apostrofami. Procedur i funkcji modułu system jest bardzo wiele. Wszystkie zostały zestawione w jednej tabeli na końcu tego dokumentu.

Po napisaniu programu na pewno chcielibyśmy zobaczyć go w akcji. Jednak każdy program przed uruchomieniem musi zostać poddany kompilacji. W TP mamy możliwość dokonania zarówno kompilacji, jak i uruchomienia. Obie te czynności wykonuje program po wybraniu polecenia Run z górnego menu o tej samej nazwie. Można to również osiągnąć kombinacją klawiszy Ctrl+F9. Jeżeli uruchomimy nasz program w taki sposób, to przez ekran przemknie kilka okienek, po czym znów zobaczymy kod źródłowy naszego programu w oknie edytora. Aby zobaczyć efekty naszej pracy należy nacisnąć kombinację klawiszy Alt+F5. Aby powrócić do ekranu edytora naciśnij Esc lub jeszcze raz naciśnij Alt+F5. Aby wyodrębnić proces kompilacji (nie uruchamiamy programu) należy wybrać z górnego menu Compile opcje Compile (nie to nie błąd) lub skorzystać z kombinacji klawiszy Alt+F9.

6. Obliczenia

Do uzyskiwania wyników obliczeń posługiwać będziemy się już znaną wam procedurą *write* oraz bardzo podobną do niej *writeln*. Różnica pomiędzy nimi polega na tym, że ta druga po napisaniu tekstu przenosi kursor do następnej linii.
Do budowania wyrażeń potrzebne nam będą operatory (tzw. znaki np. dodawanie, mnożenie itp.):

+ dodawanie
- odejmowanie
* mnożenie
/ dzielenie

Jak wiadomo najszybciej uczymy się przez praktykę. Napiszmy więc program, który pomoże nam lepiej zrozumieć ten problem.

```
Program operatory;            {nazwa programu         }
uses crt;                     {deklaracje modułu crt  }
```

```
begin                                {początek programu      }
     TextBackground(7);              {ustawiamy kolor tła    }
     TextColor(5);                   {... i kolor tekstu     }
     ClrScr;                         {czyścimy ekran         }
     Writeln('32+65=',32+65);        {dodajemy               }
     Writeln('56-34=',56-34);        {odejmujemy             }
     Writeln('83*32=',83*32);        {mnożymy                }
     Writeln('94:32=',94/32);        {i dzielimy             }
End.
```

Dyskusja:

Na początek procedura *TextBackground* zmienia kolor tła i tekstu. Następnie następuje wyczyszczenie ekranu (ekran zmienia kolor na ten ustawiony w procedurze *TextBackground*). Procedura *writeln* pisze na ekranie dokładnie to co podane zostało w apostrofach, następnie, po przecinku, wynik działania arytmetycznego. Efektem działania takiego programu będzie obraz:

```
32+65=97
56-34=22
83*32=2656
94:32= 2.9375000000E+00
```

Ostatni wynik podany jest w postaci wykładniczej – 2.9375000000E+00 odpowiada 2.9375.

Dodatkowo możemy zastosować nowy funkcje:

- `div` – całość z dzielenia (wynikiem jest część całkowita z dzielenia dwóch liczb, np. 9 div 4 = 2)
- `mod` – reszta z dzielenia (wynikiem jest reszta z dzielenia pierwszej liczby przez drugą, np. 9 mod 4 = 1)
- `abs(x)`- wartość bezwzględna z X (np. asm(-30)=30)
- `Sqr(x)` – kwadrat liczby X (np. sqr(3)=9)
- `Sqrt(x)` - pierwiastek kwadratowy z nieujemnej liczby X (np. sqrt(9)=3)
- `Sin(x)` - wartość funkcji sinus dla argumentu X, podawanego w radianach (np. sin(1)=0.84147098481)
- `Cos(x)`- wartość funkcji cosinus dla argumentu X, podawanego w radianach (np. cos(1)=0.54030230587)
- `Trunc(x)` – ilość całości w ułamku dziesiętnym (np. trunc(3.684685473)=3)
- `Round(x)` – zaokrąglenie liczby X (np. round(3.684685473)=4)

W TP nie istnieją funkcje tangens i cotangens, więc należy korzystać z zależności trygonometrycznych.

7. *Zmienne*

Obrazowo: zmienne są to „szufladki” w których przechowywane są dane. Aby posłużyć się taką „szufladką” musimy ją najpierw zadeklarować. Zmienną definiujemy po słowie *var* (ang. Variables – zmienne), nadajemy jej wymyśloną przez nas nazwę, a następnie po dwukropku określamy jej typ. Wygląda to mniej więcej tak:

```
var zmienna:typ;
```

Nazwa zmiennej nie może zawierać polskich liter (np. ąćź), nie może rozpoczynać się cyfrą oraz nie może być słowem języka Pascal (np. nazwa procedury).
Typy zmiennych języka TP:

Tabela 2

Typ	Opis
real – liczby rzeczywiste	
real	liczba rzeczywista od 2.9e-39 do 1.7e38
single	liczba rzeczywista od 1.5e-45 do 3.4e38
double	liczba rzeczywista od 5.0e-324 do 1.7e308
extended	liczba rzeczywista od 3.4e-4932 do 1.1e4932
comp	liczba rzeczywista do –9.2e18 do 9.2e18
integer – liczby całkowite	
byte	liczba całkowita od 0 do 255
word	liczba całkowita od 0 do 65535
shortint	liczba całkowita od –128 do 127
integer	liczba całkowita od –32768 do 32767
longint	liczba całkowita od –2147483648 do 2147483647
booleans	
boolean	zmienna wielkości byte
WordBool	zmienna wielkości word – akceptowana przez kompilator 32 (Windows)
LongBool	zmienna wielkości longint – akceptowana przez kompilator 32 (Windows)
ByteBool	zmienna wielkości byte – akceptowana przez kompilator 32 (Windows)
Inne	
Char	Pojedynczy znak ASCII
String	Ciąg znaków ASCII
File	Plik
Array	Tablica zmiennych

Możliwe jest łączenie zmiennych tego samego typu w jednej deklaracji (np. `a,b,c:integer`). Pora na mały przykładzik:

```
program zmienne;
uses crt;
var liczba1,liczba2:byte;     {deklarujemy dwie zmienne  }
    napis1:string[20];        {typu byte oraz jedną typu }
begin                              {string               }
     clrscr;
     liczba1:=34;                  {do zmiennych ...}
     liczba2:=19;                  {...przypisujemy... }
     napis1:='oto dwie liczby : ';{...wartości        }
     writeln(napis1,liczba1,liczba2);  {i piszemy     }
end.
```

Dyskusja:

Na początku programu deklarujemy zmienne: dwie liczby oraz napis o maksymalnej długości 20 liter. Dalej rozpoczyna się właściwy program, w którym do zmiennej *liczba1* przypisujemy wartość 34 (:= jest to operacja przypisania), do zmiennej

liczba2 wartość 19 oraz do zmiennej *napis1* przypisujemy ciąg liter zawarty w apostrofach. Na koniec procedurą *writeln* wypisujemy wszystkie zmienne na ekranie.

8. Instrukcja warunkowa (if ... then ...)

Instrukcja warunkowa wykonywana jest przez program tylko w momencie gdy spełniony zostanie określony warunek. Aby to łatwiej wytłumaczyć posłużę się przykładem:

```
program instrukcja_warunkowa;
uses crt;
var liczba:integer;
begin
     clrscr;
     write('prosze podac liczbe');
     readln(liczba);
     if (liczba mod 2)=0 then writeln('liczba parzysta')
          else writeln('liczba nieparzysta');
end.
```

Dyskusja:

Program najpierw czyści ekran, następnie wyświetla napis. Następna procedura (*readln*) wczytuje znaki z klawiaturze i przypisuje je do zmiennej (jednocześnie wyświetlając litery/cyfry na ekranie) po wciśnięciu entera kończy czytać i przenosi kursor do nowej linii. Istnieje również podobna do niej procedura *read*, która różni się od *readln* tym, że to pierwsze nie przenosi kursora do nowej linii (tak jak to się ma z *write* i *wirteln*). Następnym krokiem jaki wykona nasz program będzie instrukcja warunkowa. Jeżeli (*if*) *liczba mod 2*, czyli reszta z dzielenia przez dwa wyniesie 0, wtedy (*then*) program wyświetli na ekranie napis : liczba parzysta. W przeciwnym wypadku (*else*) na ekranie wyświetlony zostanie napis : liczba nieparzysta.

Budowa instrukcji *If..Then* jest następująca:

```
if warunek then działanie1 else działanie2;
```

Ostatni człon od słowa *else* można pominąć, zależy to od budowy programu. Lista pleceń w instrukcji warunkowej, jeśli nie jest jednoelementowa, powinna być ograniczona słowami *Begin* i *End;* po *end* nie stawiamy kropki, np.

```
if i>0 then
     begin
          polecenie1;
          polecenie2;
          polecenie3;
     end;
```

Jeśli chcemy użyć dwóch warunków należy oddzielić je jednym ze słów:

- *and* – suma – działanie jest wykonywane w przypadku spełnienia obu warunków
- *or* – alternatywa – działanie jest wykonywane w przypadku spełnienia jednego z warunków
- *xor* – różnica symetryczna argumentów – działa podobnie do *or*.

np.

```
if warunek1 and warunek2 then działanie;
if warunek1 or warunek2 then działanie;
```

W przypadku gdy chcemy aby działanie było wykonane w przypadku nie spełniania warunku należ go poprzedzić słowem *not*, np.

```
if not warunek1 then działanie;
```

Przy budowie warunków bardzo przydatne są operatory relacji. Oto lista tych, którymi możemy posłużyć się w TP:

Tabela 3

Operator	Opis
=	równe
<>	nierówne
<	mniejsze
>	większe
<=	mniejsze lub równe
>=	większe lub równe

9. *Iteracje*

Iteracje czyli mówiąc prościej pętle. Pętle zastępują na wielokrotne przepisywanie tych samych procedur. Wyróżniamy trzy rodzaje pętli:

Uwaga: Proszę zapamiętać sobie kombinacje klawiszy CTRL + PAUSE BREAK. Uratuje ona wam skórę w przypadku gdy program się zapętli.

1. Pętla z licznikiem

```
For licznik:=wart_pocz to wart_koncowa do lista_polecen
```

W pętli z licznikiem właśnie zmienna *licznik* spełnia rolę wskaźnika (licznika) nawrotów pętli i dlatego musi być typu *integer* (liczby rzeczywiste, np. *integer*, *byte*, *word*). Przyrost tej zmiennej odbywa się przez dodawanie do jej wartości 1 przy każdym „obrocie” pętli. Pętla kończy się gdy licznik równa się wartości końcowej (*wart_koncowa*). Przy budowaniu tej pętli należy pamiętać aby wartość początkowa była mniejsza od wartości końcowej. Przyszła pora na przykładzik:

```
program petla1;
uses crt;
var i:byte;
begin
     clrscr;
     for i:=0 to 15 do
          begin
               textcolor(i);
               writeln(i);
          end;
end.
```

Dyskusja:

Powyższy program przedstawia działanie pętli *for..to..do*. Program ustawia kolor tekstu, a następnie wyświetla numer koloru na ekranie.

```
For licznik:=wart_pocz to wart_kon downto lista_polecen
```

Oto drugi rodzaj pętli z licznikiem. Działa odwrotnie do opisanej wcześniej. Zamiast dodawać do wartości początkowej odejmuje od niej 1, dlatego w tej pętli wartość początkowa musi być większa od wartości końcowej. Przykład:

```
program petla2;
uses crt;
var i:byte;
begin
     clrscr;
     for i:=10 to 0 downto
          begin
               writeln(i);
               delay(1000);
               clrscr;
          end;
end.
```

Dyskusja:

Program który odlicza 10 sekund. Zastosowano w nim procedurę *delay*, która zatrzymuje działanie programu na określoną liczbę milisekund.

2. Pętla z warunkiem sprawdzanym po wykonaniu poleceń (repeat..until)

```
repeat lista_polecen until warunek
```

Pętla ta jest wykonywana tak długo aż nie zostanie spełniony warunek. W konstrukcji tej pętli słowa *repeat* i *until* spełniają rolę ograniczników dla listy poleceń i dlatego nie musimy jej zamykać w „nawiasach” *begin ... end*. No i oczywiści przykładzik:

```
program petla3;
uses Crt;
begin
     ClrScr;
     repeat
          write(random(9));
     until keypressed;
     ClrScr;
end.
```

Dyskusja:

W powyższym programie zauważamy dwie nowe funkcje:

- *random(x)* – funkcja ta zwraca losową liczbę z zakresy od zera do liczby X
- *keypressed* – funkcja ta przyjmuje wartość *true* (warunek spełniony) gdy zostanie naciśnięty klawisz na klawiaturze.

Efektem działania tego programu jest wyświetlanie na ekranie losowych liczb do czasu gdy zostanie naciśnięty klawisz.

3. Pętla z warunkiem na początku.

```
while warunek do lista-polecen
```

Pętla ta jest wykonywana w przypadku, gdy zostanie spełniony warunek. W konstrukcji tej pętli listę poleceń musimy zamykać w „nawiasach” *begin ... end*. Przykładzik:

```
Program petla4;
uses Crt;
var i:byte;
begin
     clrscr;
     n:=20;
     while n>0 do
          begin
               writeln(n);
               delay(1000);
               n:=n-1;
               clrscr;
          end;
end;
```

Dyskusja:

Ten program jest bardzo podobny do programu `petla2.` Efektem jego wykonanie jest odliczenie 10 sekund.

Różnica pomiędzy pętla `repeat...until` a `while...do` jest następująca. W tej pierwszej lista poleceń zostanie wykonana przynajmniej jeden raz, natomiast w tej ostatniej lista poleceń zostanie wykonania tylko w przypadku spełnienia warunku.