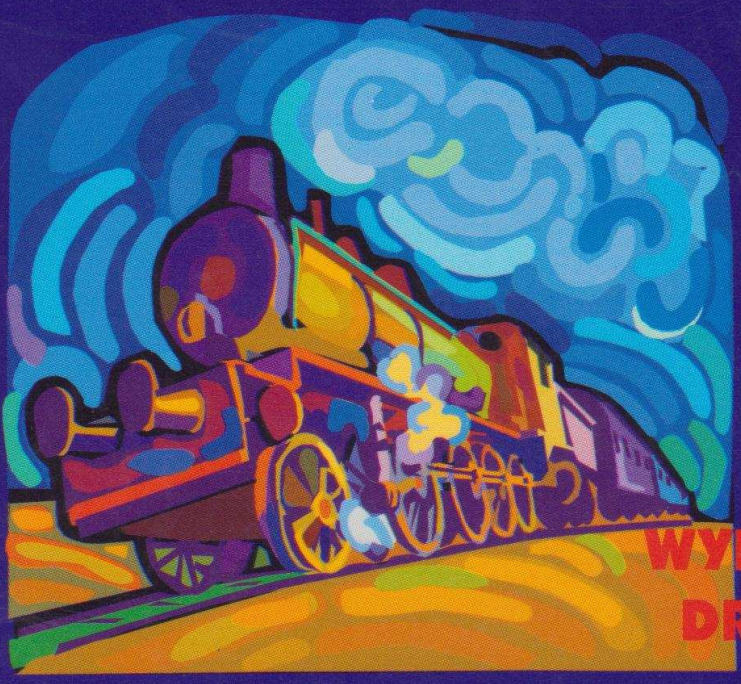
TURBO PASCAL

TWÓJ PIERWSZY PROGRAM



WYDANIE DRUGIE

Książki komputerowe od 1989 roku
HELP
www.besthelp.pl

Karol Wierzchołowski

Karol Wierzchołowski

Turbo Pascal

Twój pierwszy program

Wydanie drugie

Przygotowano na podstawie Borland/Turbo Pascal 7

Komputerowa Oficyna Wydawnicza „HELP"

I

Redakcja: *Agnieszka Michalska*

Jest to książka dla zupełnie początkujących. Założyliśmy, że czytelnik nigdy w życiu nie programował w żadnym języku. Wymagana jest tylko fundamentalna wiedza z zakresu obsługi komputera, czy systemu operacyjnego. Jeśli jesteś uczniem i uczysz się programowania na lekcjach informatyki - ta książka jest właśnie dla Ciebie. Znajdziesz w niej (lub dzięki niej) wiele ciekawych informacji, wiele zabawy i wiele prawdziwej satysfakcji. Książka i materiał w niej zawarty nie jest jednak zupełnie prosty. Wielokrotnie używam sformułowań, które mogą być dla Ciebie chwilami zagadkowe, trudne. Będę to robił celowo, abyś otarł się trochę o żargon informatyczny i programistyczny. Powodzenia!

Wydawca:
Komputerowa Oficyna Wydawnicza **„HELP",** Piotr Gomoliński
ul. Dworcowa 8
05-816 Michałowice (pod Warszawą)
tel./faks (0-22) **723** 89 21, **723** 87 64, tel. awaryjne 0-604 298 099, 0-602 633 452, 0-602 248 550
poczta elektroniczna: **piotr@besthelp.pl**
www.besthelp.pl

Druk: A-Z DRUK Raszyn, ul. Słowikowskiego 21c, tel 720 35 61

Oprawa: Raszyn, ul. Cicha 45, tel 0-601 22 37 20

ISBN 83-89391-54-6

Spis treści

Przedmowa

Mamy już XXI wiek - wiek komputerów i elektroniki. Postęp cywilizacyjny jest bardzo widoczny. Jeszcze zupełnie niedawno, standardem były procesory taktowane zegarami 300-400 MHz, czy „podkręcane" Celerony osiągające zawrotne prędkości 700 MHz. Z czym mamy do czynienia teraz? Przekroczono już kilka GHz! Parę lat temu popularne były dyski twarde o wielkich pojemnościach sięgających kilku GB. Dzisiaj standardem jest 80 GB. Dobrze pamiętam, jakim wydarzeniem było ukazanie się Windows 95. Prawdziwe „okienka"! Dzisiaj króluje Windows XP. Co będzie za rok? Za dwa? Chyba nikt nie jest w stanie tego przewidzieć.

Komputery stają się coraz szybsze i coraz bardziej powszechne. Kiedyś lekcje informatyki prowadzone były tylko na studiach. W dzisiejszych czasach taki przedmiot jest i w liceum, i w gimnazjum i - nawet - w szkole podstawowej. W każdym urzędzie, w każdej firmie, w każdej instytucji często znajduje się kilka komputerów. Zazwyczaj połączone są one w sieć pozwalającą na wymianę informacji między sobą. Coraz częściej podłączone są także do Internetu, globalnej sieci informatycznej.

Rosną zasoby sprzętowe, a wraz z nimi potrzeby programowe. Pojawiają się super szybkie karty grafiki, a za nimi gry maksymalnie wykorzystujące nowe możliwości. Trudno jest być „na czasie". Cały czas należałoby inwestować w swój komputer, aby móc uruchamiać na nim najnowsze oprogramowanie i swobodnie je wykorzystywać. Poczciwy Windows 95 po instalacji zabierał koło 30 MB. System Windows XP potrzebuje już kilkaset MB. Mimo wszystko instalujemy nowe programy i gry, a potem korzystamy z nich czy to dla zabawy, czy leż służbowo. Zachwycamy się ich możliwościami. Można by kiedyś postarać się obliczyć, jak wiele pracy wykonują za nas komputery. Ile czasu dzięki nim oszczędzamy.

Tylko czasami człowiek sobie myśli: „Jak to w zasadzie działa?". Włącza komputer, a na ekranie monitora widzi różne napisy, tabelki, rysunki. Nagle pokazuje się komunikat „Uruchamianie systemu..." i jego oczom ukazuje się pulpit. Poruszając myszką, jednocześnie porusza kursorem na ekranie. Klikając w ikonki uruchamia kolejne aplikacje, które wykonują za niego tyle pracy. Zastanawia się, czym jest tak w zasadzie komputer i jak on działa? Jak działają programy? Jak to się dzieje, że są one takie szybkie? Dlaczego niektórzy mówią, że komputer się nigdy nie myli, a przecież wiemy dobrze, że wszyscy się mylimy? Jak to się dzieje, że na niewielkiej płycie kompaktowej można zapisać film, czy kilkadziesiąt godzin muzyki? Cóż to jest kompresja danych? Jak można

skompresować plik? Jak go ścisnąć i zapisać na płycie? Cóż to za choroba te wirusy, o których czasem słyszymy w wiadomościach? I jeszcze tysiące innych pytań i wątpliwości może zaprzątać nasz umysł. Na szczęście, po chwili o tym zapominamy, bo mamy wiele innych, znacznie poważniejszych spraw na głowie.

Na wiele z powyższych pytań odnajdziesz odpowiedź w tej niewielkiej książce. Nauczymy się pisać własne aplikacje, a więc ,,rozmawiać" z poszczególnymi częściami naszego komputera, z procesorem, pamięcią, kartą graficzną. Tworząc własne programy poznawać będziemy budowę tych elementów i zasady ich działania. Oczywiście nie poznamy technicznych szczegółów budowy procesora, czy też struktury zapisu danych na różnych nośnikach, bo to nas w ogóle nie interesuje (nie na tym etapie). Poznamy podstawy, ale nie zwykłego poruszania się po „okienkach", lecz własnego tworzenia prawdziwych aplikacji.

Po przestudiowaniu tej książki, po wykonaniu ćwiczeń w niej zawartych, będziesz jeszcze bardziej zafascynowany swoim komputerem, drogi Czytelniku. Będziesz również dobrze wiedział, czym są programy komputerowe, jak się je pisze i dlaczego zawierają błędy. Nie będziesz w stanie stworzyć każdego programu, ale z pewnością będziesz potrafił usprawnić niektóre z zadań, jakie wykonujesz przy pomocy tej bezdusznej maszyny. Będziesz również mógł powiedzieć, że potrafisz programować komputery, że jesteś programistą- bo nim rzeczywiście będziesz.

"Turbo Pascal. Twój pierwszy program", to książka dla zupełnie początkujących. Założeniem jest, że czytelnik nigdy w życiu nie programował w żadnym języku. Wymagana jest tylko fundamentalna wiedza z zakresu obsługi komputera, czy systemu operacyjnego. Jeśli jesteś uczniem i uczysz się programowania na lekcjach informatyki - ta książka jest właśnie dla Ciebie. Znajdziesz w niej (lub dzięki niej) wiele ciekawych informacji, wiele zabawy i wiele prawdziwej satysfakcji. Książka i materiał w niej zawarty nie jest jednak zupełnie prosty. Wielokrotnie używam sformułowań, które mogą być dla Ciebie chwilami zagadkowe, trudne. Będę to robił celowo, abyś obył się trochę z żargonem informatycznym i programistycznym. Abyś osłuchał się (oczywiście czytając) z niektórymi pojęciami, które z pewnością dałoby się nazwać w prostszy sposób. Jednak dzięki temu rozmawiając z innymi osobami, które również programują, znacznie łatwiej się porozumiecie. Będzie mniej nieścisłości. I gwarantuję, że nie będziesz żałował.

Zapraszam do wspólnej nauki!

Karol Wierzchołowski

1. Turbo Pascal

W tym rozdziale dowiemy się na czym polega istota programowania komputerów. Czym się charakteryzuje język Turbo Pascal, którego będziemy się wspólnie uczyć. Zajmiemy się także przygotowaniem sobie środowiska do pracy, a więc instalacją i konfiguracją Pascala. Dowiemy się, czym jest kompilator, program wykonywalny i poznamy krótką historię komputerów.

1.1. Ewolucja komputerów

Pierwsze komputery w niczym nie przypominały tych, do których jesteśmy dzisiaj przyzwyczajeni. Zajmowały dużo miejsca, składały się z tysięcy lamp elektronowych, potrzebowały olbrzymich ilości energii elektrycznej. Pomimo tych wymagań nie potrafiły wykonać zbyt wielu operacji i były bardzo zawodne. Przełom nastąpił, kiedy wymyślono półprzewodniki, a następnie tranzystory.

Bezduszna maszyna jest urządzeniem, które działa w określony, zaprojektowany przez twórców sposób. Przykładowo: pobiera pewne informacje od użytkownika (liczby A i B), dokonuje na nich obliczeń i na wyjściu daje wynik (np. sumę A+B). Bardzo trudno byłoby „nauczyć" takie urządzenie rozpoznawania wszystkich 10 cyfr (choć pierwsze komputery tak właśnie działały). Odpowiedniego łączenia ich, by tworzyły liczby, operowania na nich. Znacznie łatwiej jest ją nauczyć tylko dwóch cyfr. Tak też się stało w praktyce. Komputery - jako układy cyfrowe - operują na tzw. systemie binarnym, inaczej zwanym dwójkowym. Pobierają pewne dane wejściowe, które są ciągami tylko dwóch cyfr: zer i jedynek. Przetwarzają je i na wyjściu wyprowadzają również ciągi zer i jedynek. Okazuje się bowiem, że za pomocą tylko tych dwóch cyfr można zapisać niemalże każdą liczbę. A jak widzimy na ekranach swoich monitorów - tymi dwoma cyframi można również zapisać ciekawe gry, programy narzędziowe, muzykę.

Komputer zna dwie cyfry i interpretuje je jako: „płynie prąd" (czyli 1) i „nie płynie" (0) (oczywiście w dużym uproszczeniu).Takie przerywane ciągi impulsów odpowiednio rozumie i interpretuje. Jednak, co to dokładnie oznacza? Jak to się dzieje? Czym jest właściwie komputer?

Każdy PeCet (czyli nasz komputer domowy) składa się z tzw. płyty głównej umieszczonej wewnątrz obudowy. Na owej płycie umieszczony jest procesor. Są tam również inne układy elektroniczne, np. pamięć RAM, BIOS, różne kontrolery, itd. Są także tzw. karty rozszerzeń. W owe karty możemy wkładać dodatkowe urządzenia, np. karty graficzne, karty sieciowe, czy też karty TV. Każda z takich kart - jak również sama płyta główna - posiadają pewne wyprowadzenia, do których możemy podłączyć inne urządzenia, np. klawiaturę, myszkę monitor, skaner, drukarkę, kamerę, itd. Wszystko to razem tworzy komputer.

Bez wątpienia, najbardziej inteligentnym układem jest procesor. To niewielka kostka, układ cyfrowy o wysokiej skali integracji (co znaczy, że na niewielkiej jego powierzchni umieszczono miliony tranzystorów). Zawiera wiele nóżek, którymi odbiera różne sygnały od innych urządzeń i którymi przesyła wyniki swoich operacji. Procesor potrafi bardzo szybko dodawać liczby zapisane w systemie dwójkowym (czyli po prostu swoimi nóżkami odbiera ciągi impulsów - płynie prąd i nie płynie - interpretuje je jako liczby binarne, wykonuje działanie, np. dodawanie, i na inne nóżki wyprowadza wynik operacji również w postaci binarnej). Oprócz dodawania potrafi wykonać jeszcze kilkanaście innych operacji. Potrafi odejmować, mnożyć, dzielić, wysyłać odpowiednie informacje do innych podzespołów, na przykład do swojej pamięci.

Procesor jest specyficznym urządzeniem. Innym niż wiele z tych, które nas otaczają. Przykładowo - zegarek mierzy czas i wyświetla go na specjalnym wyświetlaczu. Niczego innego nie potrafi. Kalkulator potrafi wykonać kilka prostych (lub bardziej zaawansowanych) operacji matematycznych. Nie potrafi jednak mierzyć czasu. Telewizor umożliwia oglądanie różnych programów telewizyjnych, radio - słuchanie stacji radiowych, magnetofon - słuchanie muzyki z kaset. Każde z tych urządzeń potrafi wykonać jedno lub kilka określonych czynności. Procesor jest trochę inny. Nie wykonuje on w gruncie rzeczy niczego ..sam z siebie". Potrafi oczywiście obliczyć podstawowe działania matematyczne i przesiać na swoje nóżki wyniki tych operacji, itd. Potrafi porównywać ze sobą liczby. Stwierdzać, która jest większa, a która mniejsza. Jednak nic pozwala ani na słuchanie muzyki, ani na działanie jako kalkulator (nie ma klawiatury, ani wyświetlacza), itd. Jednak w połączeniu z dodatkowymi elementami jest w stanie zastąpić prawie każde inne urządzenie elektroniczne. Trzeba mu tylko dać specjalny program, czyli ciąg prostych instrukcji, jakie ma wykonać. Taki ciąg instrukcji jest szeregiem zer i jedynek (czyli impulsów prąd płynie i nie płynie) podanych na odpowiednie nóżki. Procesor rozpoznaje wybrane sekwencje z takich danych i odpowiednio je interpretuje. Przykładowo: po ,,usłyszeniu" ciągu 01101101 (co dla niego może oznaczać „wykonaj dodawanie") będzie czekał, aż podane mu zostaną następne ciągi określające pewne liczby. Kiedy to nastąpi, na swoim wyjściu powstanie inna sekwencja, np. 011011110111, która określa jaki jest wynik poprzedniej operacji.

Spójrzmy teraz na to całościowo. Każdy komputer posiada płytę główną, na której umieszczony jest procesor, BIOS, pamięć RAM. Pamięć RAM pozwala na przechowywania olbrzymich ilości sekwencji zer i jedynek. W BlOS-ie zapisany jest (za pomocą zer i jedynek), specjalny program, czyli ciągi różnych instrukcji. Wszystkie te podzespoły są ze sobą połączone. Kiedy uruchomisz komputer program, który znajduje się w BIOS-ie zostanie wykonany, tzn. zostanie wysłany do procesora. Ten wykonując go, dokona czynności testujących sprzęt, itp. Procesor będzie wysyłał pewne sygnały (wyniki swojego programu) do karty graficznej. Ta będzie je odbierała, również odpowiednio interpretowała i wysyłała swoje własne wyniki do monitora. W ten sposób na ekranie zobaczysz różne napisy. Procesor będzie komunikował się również z klawiaturą „czytając", co jest na niej pisane. Za pośrednictwem innych urządzeń (kontrolera) będzie potrafił skomunikować się z dyskiem twardym, stacją dyskietek, CD-ROMem, itd.

Jak więc widzisz komputer to cały zestaw wielu różnych urządzeń, które są ze sobą połączone i potrafią się ze sobą komunikować. W takiej komunikacji posługują się tylko zerami i jedynkami, a więc impulsami „płynie prąd" i „nie płynie". W ten sposób - pomimo, iż procesor sam nie może wykonać zbyt wielu operacji - potrafi zrobić bardzo dużo, zlecając poszczególne czynności innym podzespołom. Widzisz teraz także, że procesor ma możliwość wpływania na inne urządzenia. Dlatego chcąc wyświetlić jakiś napis na monitorze nie komunikujemy się bezpośrednio z kartą graficzną (nie każemy jej wyświetlać, choć jest to oczywiście możliwe), a tym bardziej z monitorem, tylko poprzez odpowiednie rozkazy procesora każemy mu, aby on sam się z nią porozumiał i wysłał jej nasze żądanie. W ten sposób my, jako programiści, czyli osoby układające program, który ma zostać wykonany, możemy - „rozmawiając" w samym tylko procesorem - komunikować się także z każdym innym sprzętem, jaki jest podłączony do naszego komputera.

Eniac był jednym z pierwszych komputerów skonstruowany przez J.P. Eckerta i J.W. Mauchly'ego na Uniwersytecie Pensylwanii w roku 1943-45 na potrzeby wojska. Operował na liczbach dziesiętnych. Składał się z 42 szaf. Zawierał 18800 lamp elektronowych, 6000 komutatorów, 1500 przekaźników, 50000 rezystorów. Ważył 30 ton i pobierał około 140 kW energii w ciągu godziny. Dodanie do siebie 5000 liczb zajmowało mu około sekundy. Jego głównymi zadaniami były obliczenia tablic balistycznych, prace nad bronią jądrową i prognozowaniem pogody, a nawet obliczaniem liczby Pi. Części tego komputera są przechowywane w Smithsonian Institution w Waszyngtonie.

1.2. Istota programowania

Istota programowania polega na wysyłaniu do procesora ciągu zer i jedynek, będących rozkazami, jakie procesor ma wykonać. Wszystkich możliwych rozkazów nie jest dużo, jednak w przypadku nowych układów można je liczyć już w setkach.

Takie programowanie, o którym wspomniałem, określamy mianem programowania maszynowego. Jest ono bardzo trudne. Trudno byłoby bowiem zapamiętać te wszystkie rozkazy, składające się ze sporej ilości samych zer i jedynek. Łatwo byłoby się również pomylić wpisując w jakimś miejscu jeden zamiast zera. Dla nas taka mała pomyłka wiele różnicy nie zrobi, jednak procesor już nie zrozumie naszego rozkazu i program może nie działać poprawnie. Na początku ery komputerów tak to jednak wyglądało. Programowaniem zajmowali się nieliczni, specjalnie wyszkoleni fachowcy}', na czym oczywiście zarabiali niemało.

Pierwsi programiści zauważyli jednak niedoskonałości takiego stylu programowania i opracowali nowy. specjalny język. Nazwany został asemblerem. Był to zestaw słów, zazwyczaj trzyliterowych, kojarzących się z instrukcjami, jakie miał wykonać procesor. Przykładowo instrukcja ADD służyła do dodawania. Nie trzeba było znać jej zapisu dwójkowego (ciągu zer i jedynek). Wystarczyło samo słowo ADD. Programy, które miał wykonać procesor, były pisane w takim języku. Jednak oczywiście procesor nie był w stanie ich zrozumieć. On nadal oczekiwał tylko ciągów binarnych. Dlatego ci sami programiści napisali (już w języku maszynowym, czyli "po staremu") specjalny program, tłumacz. Owy tłumacz, zamieniał ich słowa z języka asembler (czyli np. słowo ADD). na odpowiedniki w języku maszynowym (np. 0110111101). Był to więc specjalny translator. To również spory krok w ewolucji programowania!

Jednak i takie tworzenie aplikacji nastręczało wielu problemów. Wyświetlenie prostego napisu, czy pobranie jednego znaku z klawiatury zajmowało kilka komend asemblera. Prost}' kalkulator składał się z setek linii. Wiadomym jest. że im więcej linii w programie, tym więcej błędów może w nim się kryć. Trwały więc prace nad kolejnymi językami. 1 takie też powstawały. Jednym z nich był (i jest nadal) język Pascal, opracowany w 1968r. przez Niklausa Wirtha na uniwersytecie w Zurychu i jego rozszerzenie opracowane przez firmę Borland, mianowicie Turbo Pascal.

1.3. Język Turbo Pascal

Język Pascal to kolejny krok w ewolucji języków programowania. On również składa się ze specjalnych słów oraz pewnej gramatyki (jak każdy język). Każdy program, który w nim napiszemy musimy przetłumaczyć na język maszynowy, aby móc go uruchomić (jak w przypadku asemblera). Takie programy tłumaczące nazywamy kompilatorami.

Pascal znacznie uprościł pisanie własnych programów. Wyświetlenie napisu zostało skrócone do jednego słowa - polecenia. Podobnie wiele innych często wykorzystywanych operacji zostało skróconych do pojedynczego słowa języka. Firma Borland rozszerzyła jakby zasób tych słów wprowadzając dodatkowe, własne instrukcje. Miało to oczywiście na celu jeszcze większe usprawnienie procesu projektowania nowych aplikacji. Udało się to jej wyśmienicie. Składnia i logika pozostała jednak kontynuowana ze „starego" Pascala.

Język programowania ma wiele wspólnego z językiem naturalnym. Tworzenie własnego programu to jakby pisanie wypracowania w innym języku. Musimy przestrzegać pewnych zasad gramatycznych, używać obcych sobie słów, itd. Są jednak dwie podstawowe różnice. Jedna na plus dla nas (osób programujących) i jedna na minus (ale z pozoru). Plusem jest to, że w języku programowania występuje niewielka - w porównaniu z językiem naturalnym - ilość słów. Policzyć je można w dziesiątkach, czy setkach. W każdym ze swoich programów nie użyjesz ich jednak więcej niż 50. Wyobrażasz sobie porozumienie się z osobą z innego kraju tylko za pomocą 50 słów? Za wadę można uznać to, że popełnienie najmniejszego błędu gramatycznego powoduje przerwanie procesu kompilacji (czyli tłumaczenia na język maszynowy), a to wiąże się oczywiście z faktem, że nie możemy uruchomić programu. Jest to jednak wada pozorna. Bowiem kompilator wymusza na nas pewną dokładność. Musimy być uważni i bezbłędni. Owocuje to dobrym porozumieniem się z procesorem. Rozmawiając z obcokrajowcem „dogadamy się" mówiąc z błędami (np. źle odmieniając), jednak może to być źródłem wielu problemów, niedomówień i nieścisłości. Na pocieszenie można jeszcze dodać, że kompilator i edytor Turbo Pascala bardzo pomagają w programowaniu kolorując odpowiednio składnię, wyświetlając różne komunikaty informujące o błędach, jednocześnie - bardzo często - wskazując dokładne miejsce ich wystąpienia. Również składnia samego języka, a więc i gramatyka są bardzo spójne i logiczne, przez co nie powinno być problemów z ich opanowaniem.

1.4. Dlaczego Turbo Pascal

Istnieje wiele różnych języków programowania. My wybraliśmy właśnie Turbo Pascal. Co się w nim kryje takiego ciekawego, czy ważnego, że skłoniło nas właśnie do niego? Nas, czy może nauczycieli, którzy każą nam się go uczyć? Otóż jest to język idealny dla początkujących. Jego składnia, logika, struktura są niezwykle spójne. Ułatwia to w znaczny sposób naukę i tworzenie własnych, pierwszych programów. Bardzo pomaga w pojawiających się - jakże często przy nauce - błędach. Nie pozostawia nas również w tyle. Bowiem pomimo, iż sam Turbo Pascal stworzony został z myślą o DOSie. to pakiet Delphi (również firmy Borland) przeznaczony jest dla środowiska Windows i również wykorzystuje język Pascal (Object Pascal). Ucząc się teraz programowania w Turbo Pascalu będziemy mieli gruntowne podstawy, by przesiąść się w przyszłości do Delphi i zacząć pisać własne programy pod "okienka". Dzięki narzędziu „Kylix", znając Pascala (Object Pascal) stworzymy szybko własne programy nawet pod Linuxa! Nie jest to więc język archaiczny, jak niektórzy o nim mówią. Nawet sam Turbo Pascal, czyli ten pod DOSa, którego będziemy się uczyć, pozwala na stworzenie niemalże każdej aplikacji. Słowniki, proste bazy danych, ciekawe i szybkie efekty graficzne, gry - to wszystko jest możliwe do osiągnięcia dla programujących w Pascalu. Potrzebne jest tylko doświadczenie, twórcza myśl, znajomość składni i podstawowych instrukcji języka. Tego wszystkiego (mam nadzieję) nauczysz się właśnie dzięki tej książce.

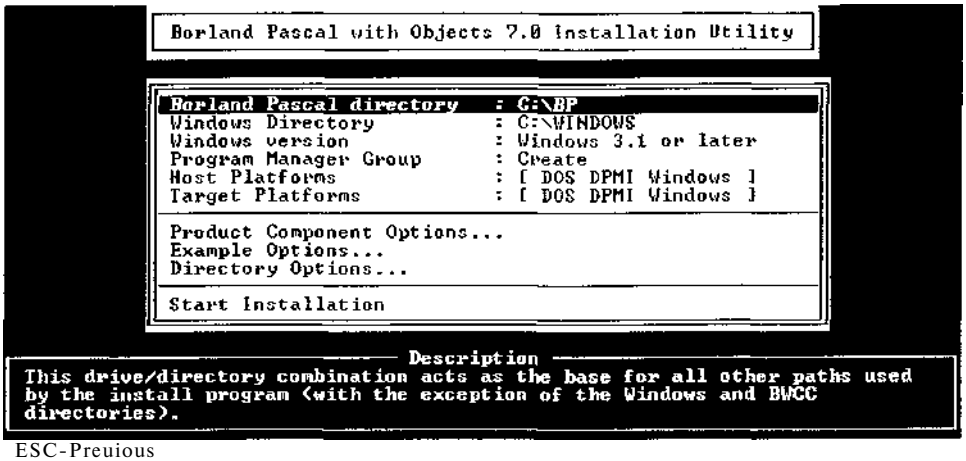
2. Środowisko Turbo Pascala

Zanim zaczniemy tworzyć własne aplikacje musimy przygotować odpowiednie środowisko pracy. Temu tematowi został poświęcony ten rozdział. Dlatego jeśli masz już zainstalowanego Turbo Pascala u siebie w domu, albo uczysz się programowania w szkole, możesz go spokojnie pominąć.

2.1. Instalacja

Instalacja pakietu Turbo/Borland Pascala jest niezwykle prosta i intuicyjna. Nikt nie powinien mieć żadnych problemów z tym zadaniem. Istnieje jednak wiele różnych dystrybucji Turbo Pascala. Przedstawię proces instalacji na podstawie wersji Borland Pascal 7.0. W starszych dystrybucjach przebieg czynności powinien być analogiczny.

Aby zainstalować pakiet włóż pierwszą dyskietkę do stacji dysków. Następnie wyświetl jej zawartość w oknie Exploratora. Odszukaj program INSTALL . EXE i dwukrotnie kliknij na nim lewym przyciskiem myszy. Ukaże się ekran powitalny. Kliknij przycisk [ENTER]. aby kontynuować. Zostaniesz zapytany o dysk źródłowy. Pozostaw wartość wpisaną wciskając ponownie [ENTER]. Zostaniesz wówczas zapytany o katalog źródłowy. Również pozostaw wpisaną wartość potwierdzając klawiszem [ENTER] . Ukaże się ekran jak na rysunku poniżej.



Rysunek 3.1. Instalacja Borland Pascala

Możemy w tej chwili wybrać wiele różnych opcji instalacyjnych, np. aby nie instalowały się pakiety pod Windows (bo nie będziemy programować pod okienka), lub aby nie instalowały się biblioteki obiektowe (gdyż z tego również nie będziemy korzystać). W tej chwili można również zmienić miejsce instalacji pakietu (domyślnie c:\bp). My jednak pozostaniemy przy ustawieniach domyślnych. Wybierając z menu Start Installation rozpocznijmy proces instalacji.

Może to trochę potrwać, w zależności od szybkości Twojego komputera. Możesz być także proszony o włożenie kolejnych dyskietek. Kiedy proces się zakończy ujrzysz ekran z dokumentacją programu. Po jej przejrzeniu zamknij okno instalatora.

Jeśli posiadasz bardzo szybki komputer, a więc Twój procesor przekracza kilkaset MHz musisz zainstalować sobie specjalnego patcha. Jeśli tego nie uczynisz niektóre z informacji i programów zawartych w tej książce nie będą działać poprawnie. Szczegółowe informacje na ten temat znajdziesz w Dodatku B

2.2. Uruchamianie

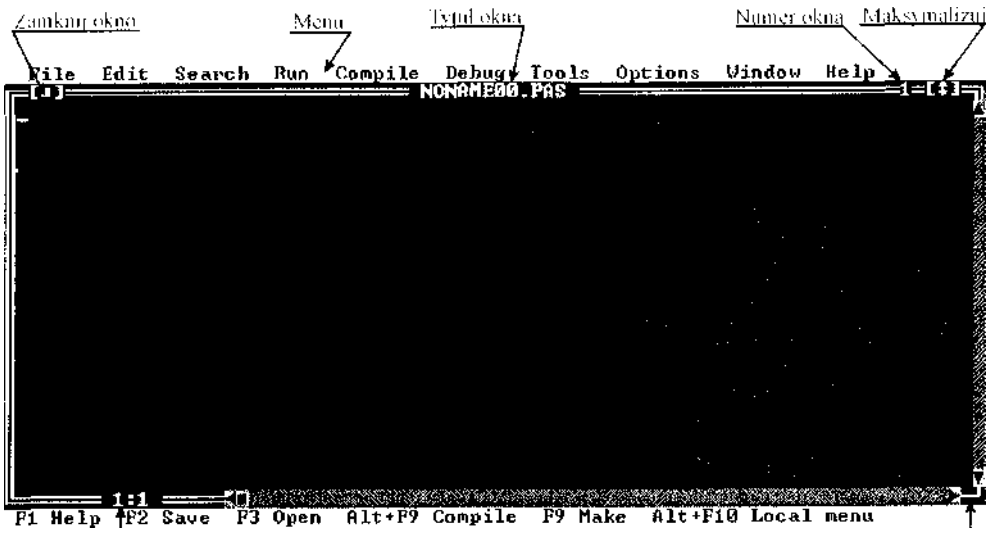
Sposób uruchomienia środowiska Turbo/Borland Pascala jest zależny od sposobu i miejsca jego instalacji, dystrybucji jaką posiadasz i wersji systemu Windows. Ponieważ zainstalowaliśmy Pascala w katalogu domyślnym jest on w c:\bp. Przejdź teraz w to miejsce, a następnie wejdź do podkatalogu bin. Odszukaj program bp.exe. Możesz stworzyć skrót do niego na swoim pulpicie, czy pasku Start, aby w przyszłości szybciej go uruchamiać. Kliknij teraz na nim dwukrotnie lewym przyciskiem myszy, aby uruchomić środowisko Borland Pascala.

BP.EXE to Borland Pascal. Jest również TURBO.EXE, czyli Turbo Pascal. Oba są oczywiście firmy Borland. BP jest jednak rozszerzeniem kompilatora Turbo Pascala. Umożliwia m.in. pracę w trybie chronionym. My z tych rozszerzeń nie będziemy korzystać, dlatego możesz swobodnie uruchamiać także TURBO.EXE, tym bardziej jeśli tak tego dokonujesz w szkole, czy na uczelni. Jeśli programujesz pod systemem Windows, możesz również uruchomić BPW.EXE, czyli wersję Borland Pascala pod „okienka".

2.3. Środowisko Turbo Pascala

Kompilator to program tłumaczący z języka zrozumiałego przez człowieka (np. Pascala, Asemblera, C, C++) na język maszynowy, czyli ciąg zer i jedynek, które potrafi wykonać (zinterpretować) procesor. Każdy program napisany w Pascalu przed uruchomieniem trzeba skompilować (przetłumaczyć). W wyniku kompilacji (i konsolidacji, czyli łączenia naszego programu z jeszcze innymi bibliotekami) powstanie plik wykonywalny, mający rozszerzenie EXE. Taki plik uruchamiamy klikając na nim dwukrotnie myszką. W rzeczywistości, kiedy tego dokonujemy, system operacyjny wczytuje go do pamięci i nakazuje procesorowi by go przeczytał i wykonał.

Istnieje wiele różnych kompilatorów do Pascala. Bardziej znane to: TMT Pascal, Free Pascal Compiler, Irie Pascal, itd. My wybraliśmy kompilator firmy Borland nie tylko dlatego, że jest szybki, posiada pewne rozszerzenia, ale w zestawie dostaliśmy całe środowisko programistyczne. Uruchamiając program BP. EXE naszym oczom ukaże się rozbudowany edytor tekstu, w którym możemy pisać nasze programy. Jednym kliknięciem klawisza na klawiaturze uruchomimy kompilator, który przetłumaczy nasz program. Gdy wystąpią błędy, oprócz komunikatu przeniesieni zostaniemy do odpowiedniej linii. Mamy rozbudowaną pomoc, pełną przykładów, zintegrowany debugger umożliwiający szybkie odszukiwanie błędów i wiele innych usprawnień. Z niektórymi zapoznamy się poznając kolejne elementy języka Pascal.



Rysunek 3.2. Środowisko Turbo Pascala

Tak wygląda ekran po uruchomieniu programu BP. EXE. Jest to środowisko programistyczne, a więc edytor z wbudowanym kompilatorem, debuggerem i systemem pomocy.

Na górze widzimy tzw. menu. Umieszczone są w nim różne opcje, które możemy wybierać myszką lub też klawiaturą. Natomiast w środkowej części ekranu widzimy niebieski ekran edytora kodu. W tym miejscu wpisywać będziemy nasze wspólne programy w języku Turbo Pascal.

W jednej chwili można pracować na wielu dokumentach jednocześnie. Wybierz z menu FILE opcję NEW. Ukaże się drugie okno edytora kodu. Każde okno edytora ma swój własny numer, który je identyfikuje. Klikając myszką w dowolny obszar okna przełączamy je na pierwszy plan. Możemy tego dokonać również samą klawiaturą wciskając klawisz [ALT+numer_okna], np. [ALT+1]

Spróbuj teraz utworzyć kilka dokumentów i w każdym wpisz dowolny tekst. Skorzystaj następnie z opcji: Copy, Paste i Cut - dostępnych w menu EDIT. Służą one do kopiowania, wklejania i wycinania zaznaczonych fragmentów tekstu do i ze schowka. Korzystając z menu FILE wybierz opcję SAVE AS, aby zapisać dokument na dysku. Z tego samego menu, opcją OPEN możesz wczytać dowolny, uprzednio zapisany, plik. Spróbuj się tym teraz pobawić.

3. Podstawy programowania

W tym rozdziale przedstawione zostaną podstawowe informacje na temat budowy i struktury programów pisanych w Turbo Pascalu. Poznamy pierwsze instrukcje tego języka. Napiszemy, skompilujemy i uruchomimy nasz pierwszy program. Dowiemy się również co to jest komentarz w programie, jak i gdzie z niego korzystać, czym są identyfikatory, z czego się składają, czym jest algorytm itp.

3.1. Pierwszy program

Uruchom środowisko Turbo Pascala. Następnie wpisz taki oto program:

```
program PO 01;
begin
   Writeln ( 'Mój pierwszy program napisany -*
             w Turbo Pascalu');
end.
```

Ze względu na to, że niektóre instrukcje łącznie z argumentami są bardzo długie i nie mieszczą się w jednej linii na łamach tej książki, przyjęliśmy taki zwyczaj, że kiedy dokonujemy „łamania" wiersza, używamy znaczka *-**. Dlatego kiedy będziesz przepisywał programy do swojego edytora, nie przenoś kursora do następnej linii, kiedy napotkasz ten znak, tylko kontynuuj wpisywanie w jednym wierszu.

Jest to tzw. kod źródłowy programu. Aby go uruchomić musimy dokonać w pierwszej kolejności kompilacji, czyli przetłumaczyć treść języka Pascal (w którym napisaliśmy powyższy program) na kod wynikowy (maszynowy), czyli taki, który procesor potrafi wykonać (zinterpretować). Środowisko Turbo Pascala posiada zintegrowany (wbudowany) kompilator, co znacznie upraszcza sprawę. Wystarczy, że wciśniesz w tej chwili klawisz [F9] lub z menu COMPILE wybierzesz opcję COMPILE. Ukaże się wówczas małe okno, w którym pokazany zostanie postęp kompilacji. Z uwagi na szybkie w dzisiejszych czasach komputery, proces ten zakończy się bardzo szybko, o czym będzie świadczył napis: *„Compile successful."*. Wówczas wciskając dowolny klawisz na klawiaturze zamkniemy to okienko.

Kompilacja się zakończyła, a więc kompilator utworzył tłumaczenie naszego programu na język zrozumiały przez procesor. Wynik tego tłumaczenia został zapisany w specjalnym pliku z rozszerzeniem . EXE, a więc takim, który system operacyjny (DOS/Windows) potrafi uruchomić (czyli nakazać procesorowi jego wykonanie).

Aby uruchomić program wciśnij przycisk [CTRL+F9] lub z menu RUN wybierz RUN. Ponieważ nasz program ma na celu tylko wyświetlenie na ekranie odpowiedniego napisu, wykona się bardzo szybko. Kiedy uruchamiamy jakiś swój program z poziomu Turbo Pascala (właśnie poprzez kombinację [CTRL+F9]) po zakończeniu działania programu następuje powrót do środowiska Pascala. Tak też się stało i tym razem. Jednak z uwagi na szybkość komputera proces ten trwał tak krótko, że być może w ogóle nie zauważyłeś, że coś się stało. Dlatego wciśnij kombinację [ALT+F5], bądź z menu DEBUG wybierz USER SCREEN. Pokazany zostanie ekran roboczy, jaki był w chwili wyłączenia Twojego programu. Zobaczysz tam, że Twój program rzeczywiście wyświetlił wskazany napis. Wciśnij dowolny klawisz, aby powrócić do edytora.

Turbo Pascal jest pewnym językiem i jak każdy prawdziwy język posiada swoje reguły, których trzeba przestrzegać, aby porozumieć się z procesorem, a wcześniej z kompilatorem. Bowiem kompilator, to taki idealny tłumacz, który jest bardzo stanowczy, jeśli chodzi o przestrzeganie zasad gramatyki, itd. Nie jest jednak "wredny", bowiem pozwala na niechlujstwo i brak estetyki, stara się również zawsze wskazać miejsca, gdzie popełniliśmy błędy. Jest bardzo cierpliwy. Jednak nigdy nie podejmie za nas decyzji i sam nie poprawi naszego błędu, nawet gdyby dokładnie wiedział, co trzeba zrobić. Dlatego miej to na uwadze.

3.2. Struktura programu

Przyjrzyj się raz jeszcze Twojemu pierwszemu programowi. Wyświetla on jeden, zwykły napis, a składa się aż z czterech linii kodu. To skutek tego, że każdy program napisany w Turbo Pascalu posiada stałą strukturę, czyli pewną określoną budowę.

Każdy program rozpoczynamy specjalnym słowem program, po którym umieszczamy nazwę naszego programu. W naszym przykładzie nazwą jest PO01. Nazwa może być oczywiście prawie dowolna. My przyjęliśmy takie nazewnictwo, aby było ono w miarę czytelne i pozwoliło na konsekwencję w kolejnych przykładach. Po nazwie programu stawiamy średnik. Cała sekwencja, aż do napotkania znaku średnika jest jedną instrukcją, którą kompilator próbuje

przetłumaczyć. Dlatego bądź uważny i nie przegap tego małego znaczka przy przepisywaniu.

Kolejny blok każdego programu to słowo `begin`, które rozpoczyna właściwy ciąg napisanych przez Ciebie instrukcji i słowo `end`, które ten ciąg kończy. Zwróć uwagę, że słowo `end` zakończone jest kropką. Tak jak zdanie kończymy kropką, tak również każdy program kończymy tym znakiem.

Pomiędzy słowami `begin` (początek) i `end` (koniec) umieszczamy instrukcje, które chcemy, aby zostały wykonane. W naszym przykładzie znajduje się tylko jedna taka instrukcja. Nazywa się `Writeln`. Służy ona właśnie do wyświetlania napisów na ekranie monitora.

3.3. Identyfikatory

Pisząc program używamy różnych instrukcji. Przykładowo: poznaną już procedurę `Writeln`, która wyświetla napisy. Jednak skąd kompilator wie, że wyświetla ona napisy? Że została poprawnie wykorzystana, itp.? Wie, ponieważ gdzieś została zapisana (my możemy tego nie widzieć) jej definicja. Definicja, tzn. ciało danej procedury, a więc ciąg mniejszych instrukcji, które mają być wykonywane, kiedy my wywołamy właśnie `Writeln`. Takich instrukcji w Turbo Pascalu jest bardzo wiele i kompilator musi pamiętać ich wszystkie nazwy. Tak też się dzieje. Jednak, aby kompilator mógł jednoznacznie rozpoznać nazwę, tzn. by dokładnie wiedział, co ma z nią zrobić, wprowadzono pewne ograniczenia. Po pierwsze - nazwy takie nie mogą się w programie powtarzać. Czyli nie mogą w programie wystąpić dwie identyczne nazwy (instrukcje) wykonujące różne zadania. Skąd bowiem kompilator miałby wiedzieć, której użyć? Po drugie - w takich nazwach możemy używać tylko małych i wielkich liter, cyfr oraz znaku podkreślenia (_). Nie możemy, więc nadać nazwy jakiejś własnej instrukcji (którą samodzielnie stworzymy) z wykorzystaniem znaków +, -, *, czy jeszcze innych (np. spacji). Nie możemy w nich również użyć polskich znaków diakrytycznych, a więc ą, ę, ś, itd. Ostatnie wymaganie dotyczy cyfr. Możemy ich użyć, ale nie na początku takiej nazwy. Dlatego nazwa nie może składać się z samych cyfr. Musi zawierać chociaż jedną literę lub podkreślenie, które umieszczone zostaną na samym początku. Poprawne nazwy mogą więc być następujące:

```
dodaj, zmienna, x, imie, ala_ma_kota, pi, a234, b3c_8
```

Poniżej wymieniłem kilka błędnych nazw:

```
12, 3zm, ala ma kota, a*b
```

Nazwy, o których piszę nazywamy identyfikatorami. I nie są to tylko nazwy instrukcji, np. Writeln, ale jeszcze inne obiekty. Przykładowo: nazwa programu, której użyliśmy w naszym pierwszym programie (PO 01) jest także identyfikatorem. Co oznacza, że nie może ona składać się z nieprzepisowych znaków. Również wszystkie nazwy zmiennych, stałych, modułów, typów danych, o których będziemy uczyć się w przyszłości - są identyfikatorami. Bowiem za ich pomocą kompilator rozpoznaje intencje programisty (Twoje) i wie jak ma przetłumaczyć kod. Dlatego staraj się zapamiętać wymagania, jakie są stawiane poprawnym identyfikatorom oraz zwróć uwagę, aby nie stworzyć w swoim programie kilku identycznych nazw.

3.4. Komentarze i wcięcia

Jeszcze raz zwróć uwagę na nasz pierwszy program. Składa się on z czterech linii, których znaczenie pokrótce zostało przeze mnie omówione. Jednak ten sam program mógłby zostać napisany również w ten sposób:

```
program P002; begin Writeln ('Mój pierwszy -*
program napisany w Pascalu');end.
```

Jak widzisz został on znacznie skrócony. Kompilator przy tłumaczeniu programu pobiera całe instrukcje (czyli sekwencje kończące się znakiem średnika) i tłumaczy je jako całość. Nie próbuje więc tłumaczyć zawsze całej linii na raz. Nie zwraca również uwagi na dodatkowe spacje, które samodzielnie wstawimy. Takie dodatkowe spacje tworzone w odpowiednich miejscach nazywamy wcięciami. W naszym pierwszym przykładzie stworzyliśmy wcięcie przed instrukcją Writeln. Nie jest ono konieczne, bo jak już wiesz kompilator je zignoruje, ale znacznie ułatwia czytanie programu (dla nas) i jego późniejszą analizę. Dlatego staraj się wszystkie swoje programy pisać w taki właśnie sposób. Obserwuj uważnie, jak ja to stosuję i próbuj robić podobnie.

Spójrz na kolejny program:

```
program PO03;
begin
     Writeln('Turbo Pascal - Samouczek');
    {ta linia wyświetli napis}
end.
```

Skompiluj go wciskając [F9], a następnie uruchom [CTRL+F9]. Aby zobaczyć wynik swojego programu wciśnij [ALT+F5]. W efekcie, na ekranie wyświetlony został napis *„Turbo Pascal - Samouczek"*. W programie umieściliśmy jednak tzw.

komentarz. Umieszczony jest pomiędzy nawiasami klamrowymi. Czy zauważyłeś, co mógł on zmienić w naszym programie? Spójrz na wynik jeszcze raz [ALT+F5]. Czy jest jakaś różnica w porównaniu do przykładu P001? Oczywiście oprócz treści komunikatu? Jak znajdziesz różnicę, to jesteś orłem. Bowiem ta sekwencja niczego nie zmieniła. Została całkowicie pominięta przez kompilator przy tłumaczeniu z Pascala na język maszynowy. Kompilator w ogóle na nią nie spojrzał.

Przed kompilacją i uruchomieniem każdego ze swoich programów nie zapomnij o jego zapisaniu na dysku. Podczas nauki popełnisz niejeden błąd i może się zdarzyć, że komputer się zawiesi i będzie potrzebny restart. Aby nie stracić swojej pracy jedno wciśnięcie klawisza [F 2] spowoduje zachowanie dokumentu.

Tak więc komentarze są to sekwencje (napisy) umieszczone w nawiasach klamrowych. (W edytorze wyświetlane są innym kolorem.) Możemy w nich wpisywać dowolne informacje, bowiem kompilator je całkowicie pomija. Mógłbyś teraz zadać pytanie - więc po co je stosować? Dla kompilatora są niepotrzebne, ale nam się przydadzą. Ucząc się kolejnych instrukcji języka będziemy mogli przy nowych procedurach czy funkcjach napisać, do czego one służą. Dzięki temu, kiedy wrócimy do naszego programy po jakimś czasie, jedno spojrzenie rozwieje wszelkie wątpliwości. Staraj się komentować swoje programy. Dzięki temu szybciej nauczysz się poszczególnych instrukcji i znacznie łatwiej będzie Ci zrozumieć pisane aplikacje.

3.5. Słowa kluczowe

W naszych pierwszych przykładach mógł przykuć Twoją uwagę kolor niektórych słów. Zauważyłeś zapewne, że słowa `program`, `begin` i `end` podświetlane są na biało, a już `Writeln` wyświetlane jest w „normalny" sposób. Mogłeś też mieć wątpliwości, kiedy pisałem, że kompilator tłumaczy całe instrukcje, a więc sekwencje zakończone średnikiem. Wynikałoby z tego, że `begin Writeln(...);` jest instrukcją jako całość. Tak jednak nie jest.

W Turbo Pascalu zdefiniowanych jest kilkanaście tzw. słów kluczowych. Wszystkie one wyświetlane są na biało i mają swoje specyficzne, konkretne znaczenie. Można je traktować jako pewne wyjątki. Same w sobie, niczego nie wykonują (z reguły). Słowo `begin` nic nie wyświetla, nic nie liczy, nic nie wykonuje. Podobnie słowo `end`, czy też `program`. Wpływają one jednak na sposób tłumaczenia programu. `Begin` mówi, że tu jest początek pewnego bloku, `end` - że

tu jest koniec bloku, program - że tutaj rozpoczyna się program i posiada on określoną nazwę, itd.

Oto lista kilku słów kluczowych, których znaczenie poznaliśmy lub poznamy w przyszłości:

- program - rozpoczyna każdy program
- begin - rozpoczyna blok programu
- end - kończy blok programu
- uses - deklaruje dodatkowe moduły
- var - deklaracja zmiennych
- const - deklaracja stałych
- array-tablica
- type - definiowanie typu danych
- if - warunek „jeśli"
- else - „w przeciwnym wypadku"

3.6. Wyświetlanie napisów

W naszym przykładowym programie wykorzystaliśmy tylko jedną instrukcję, której działanie było wyraźnie widoczne, mianowicie Writeln. Służy ona do wyświetlania napisów. Przypomnijmy jej zapis:

```
Writeln('To jest tekst');
```

Otóż instrukcje (procedury/funkcje) możemy podzielić na dwie grupy. Pierwszą z nich są te, które nie potrzebują żadnych argumentów dodatkowych, natomiast drugą grupą te, które potrzebują dodatkowych argumentów, aby zostać poprawnie wykonane. Instrukcja Writeln jest połączeniem tych dwóch grup. W naszym przykładzie wykorzystaliśmy ją z dodatkowym argumentem. Zwróć uwagę, jak jego dokonaliśmy. Zawsze argumenty do instrukcji przekazujemy w nawiasach okrągłych i -jeśli jest ich kilka - oddzielamy przecinkami. W naszym przykładzie procedurze Writeln przekazaliśmy tylko jeden argument i (jak podałem wcześniej), umieściliśmy go w nawiasach. Ten argument określa treść napisu, jaki chcemy wyświetlić. Zwróć uwagę, że jest on umieszczony w apostrofach. To bardzo ważne.

W swoim programie można użyć kilka razy instrukcji Writeln. Spójrz na przykład:

```
program PO04;
begin
   Writeln('Pierwsza linia tekstu');
   Writeln('Druga linia tekstu');
   Writeln('Trzecia linia tekstu');
end.
```

Wciśnij przycisk [F9], aby skompilować program, następnie [CTRL+F9], aby go uruchomić i [ALT+F5], aby zobaczyć jego efekt.

Kompilator tłumaczy program w takiej kolejności, w jakiej go napiszesz, a zatem i procesor wykonuje go w tej samej kolejności. Czyli od góry do dołu. Jeśli teraz ten program skompilujesz i uruchomisz zobaczysz trzy linijki wyświetlonego tekstu. Zwróć uwagę, że każdy napis umieszczony jest w nowej linii, a nie na przykład obok poprzedniego. Dokładniej precyzując - procedura Writeln wyświetla napisy w miejscu kursora (migający prostokąt), a następnie ów kursor przenosi o jedną linię w dół. W ten sposób kolejne wywołanie procedury powoduje wyświetlanie napisów jeden pod drugim.

Procedurę Writeln można jednak użyć bez podawania żadnych argumentów. Wówczas pomijamy nawiasy i treść między nimi. Takie jej wywołanie sprawia, że nie zostanie wyświetlony żaden napis, ale kursor przeniesiony zostanie do linii następnej. W ten właśnie sposób możemy robić odstępy. Spójrz na przykład:

```
program P005;
begin
   Writeln('Turbo Pascal - Samouczek');
   Writeln;
   Writeln('Przykład numer 5');
end.
```

```
Wynik:

Turbo Pascal - Samouczek

Przykład numer 5
```

Zwróć uwagę, że pomiędzy wyświetlanymi napisami jest pusta linia.

Analogiczną do procedury Writeln jest instrukcja Write. Wykorzystujemy ją w identyczny sposób. Różnica między nimi jest taka, że instrukcja Write nie przenosi kursora na koniec do kolejnej linii, lecz na koniec wyświetlanego tekstu Można więc przy jej pomocy wyświetlać napisy jeden obok drugiego. Spójrz na przykład:

```
program PO06;
begin
   Write('Jaś   ');
   Write('Kowalski  ');
   Writeln('lubi  programować.');
   Write('A to jest już w kolejnej  linii');
end.
```

Wynik:

```
Jaś Kowalski  lubi programować.
A to jest już w kolejnej  linii
```

Procedury Write nie można już wykorzystywać bez podawania argumentów. Są one konieczne. Bo cóż miałoby znaczyć samo Write? Wyświetl „nic" i pozostaw kursor w tym samym miejscu? Nie miałoby to większego sensu.

Zarówno Write jak i Writeln nie mają jawnie określonej wymaganej liczby argumentów. Może ich być zmienna ilość. Wszystkie oddzielamy od siebie przecinkami. Spójrz na przykład:

```
program  PO07;
begin
   Writeln('Napis1','Napis2','Napis3');
end.
```

Każdy z napisów zostanie wyświetlony jeden obok drugiego, bowiem dopiero na końcu kursor przeniesiony zostanie do nowej linii. Takich argumentów moglibyśmy przekazać oczywiście jeszcze więcej. Nie ma co do tego żadnych ograniczeń.

Dotychczas jako argumenty instrukcji przekazywaliśmy ciągi umieszczone w apostrofach. Każdy tekst objęty tymi znakami jest interpretowany jako napis i nie podlega żadnej dodatkowej analizie. Możemy więc w nim umieszczać dowolne znaki. Wyjątkiem jest tylko sam znak apostrofu. Spójrz na przykład:

```
program PO08;
begin
   Writeln('I''m 21 years old.');
end.
```

Wynik: I`m 21 years old.

Ponieważ napisy obejmujemy apostrofami, nie możemy ich normalnie używać wewnątrz takiego napisu. Kompilator mógłby wówczas zinterpretować taki do-

datkowy apostrof nie jako ten do wyświetlenia, a jako koniec tekstu. Rozwiązanie polega na jego dwukrotnym użyciu (jak w przykładzie).

Co się jednak stanie, jeśli w ogóle nie obejmiemy tekstu tymi znakami? Spójrz na nasz kolejny przykład:

```
program PO09;
begin
   Writeln(2+2);
end.
```

```
Wynik:
4
```

Skompiluj [F9], uruchom [CTRL+F9] i obejrzyj wynik programu [ALT+F5].

Jak widzisz nie umieściliśmy sekwencji „2+2" w apostrofach i program poprawnie się skompilował. W wyniku jego uruchomienia wyświetlona została liczba 4. Czyli poprawny wynik powyższego działania. O czym to świadczy? O tym, że wyrażenie 2+2 zostało obliczone i wynik został przekazany procedurze Writeln do wyświetlenia. Wniosek można wyciągnąć następujący. Jeśli jako argument jakieś instrukcji przekazujemy pewne wyrażenie, czy też inną instrukcję, to w pierwszej kolejności wykonana zostanie ta wewnętrzna instrukcja, a dopiero później zewnętrzna. Wrócimy do tego jeszcze za chwilę. Tymczasem pamiętaj, że instrukcje wykonywane są od środka (od najbardziej zagłębionej) do tej zewnętrznej.

Działanie 2+2 kompilator jest w stanie obliczyć. Potrafi on bowiem dodawać, odejmować, mnożyć, czy dzielić. Dlatego korzystając z tych operatorów możemy budować bardziej skomplikowane działania i przekazywać je jako argumenty instrukcji Writeln. Wówczas wyświetlony zostanie wynik operacji. Spójrz na kolejny przykład:

```
program P010;
begin
   Writeln('2+2=',2+2);
end.
```

```
Wynik:
2+2=4
```

W tym przykładzie procedurze Writeln przekazaliśmy dwa argumenty. Wyglądają one podobnie, jednak nie identycznie. Pierwszy argument przekazany został w apostrofach, a to - zgodnie z tym co napisaliśmy wcześniej - powoduje wyświetlenie danej sekwencji bez żadnej jej interpretacji. Dlatego wyświetlony

został napis „2+2=". Drugi argument jest wyrażeniem 2+2, które nie jest opatrzone apostrofami, a więc zostanie wykonane, czego efektem będzie wynik 4.

Co by się jednak stało, gdybyśmy napisali taki oto program:

```
program PO 11;
begin
   Writeln(Jakiś przykładowy napis);
    {w tym programie jest błąd -
     tekst nie jest w apostrofach}
end.
```

Spróbuj skompilować powyższy przykład wciskając [F9] . Zobaczysz komunikat o błędzie, a kursor umieszczony zostanie w linii, w której ów błąd został znaleziony.

Rysunek 4.1. Nieznany identyfikator

W taki właśnie sposób kompilator podpowiada, na czym polega usterka. Treść komunikatu brzmi: *„Unknown identifier",* to oznacza, że kompilator nie zna identyfikatora Jakiś. Wniosek jest taki, że tutaj również kompilator próbuje wykonać instrukcję Jakiś przykładowy napis, a dokładnie samego słowa Jakiś, co oczywiście jest błędem, bo nie jest to żadna instrukcja. Ale spójrz na następny przykład:

```
program P012;
begin
   Writeln('Pierwiastek z 4=',sqrt(4));
end.
```

Wynik:

```
Pierwiastek z 4 = 2.0000000E+00
```

W powyższym programie procedurze Writeln przekazaliśmy również dwa argumenty. Pierwszy jest zwykłym tekstem, który zostanie wyświetlony. Natomiast drugi to sqrt(4). Zgodnie z tym, czego już się uczyliśmy, ponieważ nie jest ta sekwencja objęta klamrami, kompilator spróbuje wykonać to działanie i nawet mu się to uda. Bowiem sqrt jest znaną kompilatorowi instrukcją, która służy do obliczania pierwiastka kwadratowego. Funkcja ta oczekuje jednego argumentu, z którego ten pierwiastek liczy i zwraca wynik swoich obliczeń. My jako argument (umieszczony w nawiasach) przekazaliśmy liczbę 4. Funkcja ta obliczy pierwiastek z tej liczby, czego wynikiem jest 2 i taką też wartość przekaże jako drugi argument procedurze Writeln. Ta z kolei wyświetli liczbę na ekranie.

Widzisz teraz, że instrukcje można dowolnie zagnieżdżać. Oto jeszcze jeden przykład:

```
program PO 13;
begin
   Writeln('Wynik=',sqrt(sqrt(4*4)));
end.
```

Spróbuj go samodzielnie przeanalizować.

Wiesz teraz, że każdy napis musi być objęty apostrofami, aby kompilator nie próbował go tłumaczyć. Chyba, że jest to świadoma czynność, ponieważ dane słowo jest znaną kompilatorowi funkcją (jak np. sqrt).

3.7. Algorytmy

Z pojęciem algorytmu spotkasz się jeszcze wielokrotnie. Zarówno jako czytelnik tej książki, jak również jako przyszły programista. Co prawda nie jest to pojęcie związane tylko z informatyką i programowaniem, ale tutaj ma szczególne znaczenie.

Każdy program, który będziesz pisał, ma wykonać jakieś określone zadanie. Nikt przecież nie pisze programów „tak sobie" bez przeznaczenia. Może to być aplikacja obliczająca pierwiastki równania kwadratowego na lekcje matematyki, rysująca wykresy różnych funkcji matematycznych, mała baza danych do przetrzymywania adresów znajomych, własny słownik ortograficzny, prosta gra, itp. Każdy z tych przykładowych programów to pewne zadanie, które jako programista musisz rozwiązać pisząc odpowiedni kod, np. w Turbo Pascalu. Jednak, jak napisać taki program? Jak napisać prostą bazę danych, aby można było przetrzymywać nazwiska, numery telefonów i adresy e-mail swoich znajomych? Jak zapisywać takie dane na dysku? Jak odszukiwać z takiej bazy określone

nazwiska? Są to pewne problemy, z którymi możesz się zetknąć pisząc określony program, czyli rozwiązując określone zadanie.

Algorytm jest to pewien przepis, sposób na rozwiązanie określonego zadania problemu). Każdy program można napisać na wiele różnych sposobów, a więc korzystać z wielu różnych algorytmów rozwiązujących dany problem. Ważne jest, aby wszystkie one wykonywały dokładnie to zadanie, które zostało postawione i aby nasza aplikacja zwracała poprawne wyniki. Mając kilka algorytmów wykonujących to samo zadanie musimy wybrać ten, który jest najlepszy do naszych zastosowań.

Przykładowo - w jednym z kolejnych rozdziałów nauczysz się przechowywać spore ilości nazwisk. Poznasz również tzw. algorytm sortowania bąbelkowego, który umożliwia poukładanie tych nazwisk w kolejności alfabetycznej. Istnieją jednak jeszcze inne metody sortowania, czyli wykonujące dokładnie to samo zadanie. Są znacznie szybsze, tzn. potrafią w jednej sekundzie poukładać większą ilość nazwisk. Jednak są bardziej skomplikowane (składają się z większej ilości linii kodu), być może potrzebują więcej pamięci komputera, itd.

Musisz jednak zapamiętać, że sposób na rozwiązanie jakiegoś problemu nazywamy algorytmem. Pisząc programy układamy i stosujemy różne algorytmy. Każdy problem da się rozwiązać na kilka różnych sposobów.

4. Stałe i zmienne

W tym rozdziale poznamy zmienne i stałe - bardzo ważne elementy każdego języka programowania. Nauczymy się również wykonywać proste operacje matematyczne na podstawie danych przekazanych przez użytkownika. Materiał z tego rozdziału jest nieco trudniejszy od tego prezentowanego dotychczas, jednak nie powinieneś mieć z nim większych problemów.

4.1. Stałe

Tworzenie stałych polega na przypisywaniu zdefiniowanym przez siebie identyfikatorom określonych wartości. Dzięki temu, zawsze kiedy w programie użyjemy swojej nazwy (owego identyfikatora) kompilator „podstawi" za niego zdefiniowaną przez nas wartość.

Tworzenie stałych nazywamy ich deklaracją i używamy w tym celu specjalnego słowa kluczowego `const`. Deklaracja nie jest instrukcją, którą wykonuje procesor. Dlatego nie umieszczamy jej w ciągu takich instrukcji pomiędzy słowami `begin` i `end`, a przed tym blokiem. Spójrz na przykład:

```
program PO14;
const x = 12;
begin
    Writeln(x*x);
end.
```

Wynik:

```
144
```

W powyższym programie zadeklarowaliśmy własną stałą. Nadaliśmy jej nazwę x i wartość 12. Stworzony więc został identyfikator o tej wartości. Dlatego kiedy dalej w programie kompilator napotka sekwencję x*x potraktuje ją jako 12*12, a to oczywiście będzie w stanie obliczyć.

Stałe mogą być również napisami, wówczas umieszczamy je w apostrofach. Może ich być także więcej niż jedna. Nie jest wtedy potrzebne wielokrotne wymienianie słowa kluczowego `const`. Wystarczy zrobić to raz, na samym początku. Kolejne deklaracje oddzielamy średnikami. Spójrz na przykładowy program:

```
program P015;
const imie = "Karol";
      Wiek = 21;
begin
    Writeln(imie,' ma ',wiek,' lat.');
end.
```

'Wynik:

```
Karol ma 21 lat.
```

Przy nazywaniu swoich stałych musisz pamiętać o wymaganiach, jakie muszą spełniać poprawne identyfikatory.

4.2. Zmienne

Gdy tworzymy własną stałą, tak w zasadzie informujemy kompilator, co ma "podstawić" pod określony, zdefiniowany przez nas, identyfikator. Przyglądając się „z zewnątrz" zmiennym zauważamy, że to również identyfikatory, w miejsce których kompilator podstawia określone wartości (a więc jak w przypadku stałych), jednak ich wartość może być wielokrotnie zmieniana.

Turbo Pascal rozróżnia liczby całkowite, liczby rzeczywiste, napisy, itd. Tworząc Własną stałą i nadając jej wartość, kompilator może samodzielnie ,.domyślić" się, jaki jest to rodzaj danych. Jeśli wpiszemy wartość stałej w apostrofach, oznaczać to będzie, że jest to napis. Jeśli wpiszemy liczbę bez ułamka - oznacza, że jest to liczba całkowita. Jeśli z ułamkiem - rzeczywista. Kiedy tworzymy zmienną- nie nadajemy jej żadnej wartości. Możemy ją, bowiem nadać w każdej chwili działania programu. Kompilator nie potrafi przewidzieć, jakie w przyszłości użytkownik będzie chciał przypisać dane do zmiennej, dlatego trzeba to jawnie określić przy deklaracji. Spójrz na przykład:

```
program P016;
var x : Integer;
begin
   X:=12;
   Writeln(x*x);
end.
```

Wynik:

Deklaracje zmiennych umieszczamy po słowie kluczowym var - także przed rozpoczęciem programu głównego (przed słowem begin). Deklaracja wygląda w ten sposób, że po nazwie (identyfikatorze) zmiennej i znaku dwukropka piszemy typ danej zmiennej i całość kończymy średnikiem. W naszym przykładzie użyliśmy zmiennej typu Integer - oznacza to, że do zmiennej x będą mogły być przypisywane tylko liczby całkowite. Zauważ jednak, że w tym momencie nie określiliśmy, co ma być pod tą zmienną. Dopiero w samym programie napisaliśmy: x:=12;. W ten sposób przypisaliśmy wartość 12 do zmiennej x. Wykorzystaliśmy w tym celu operator przypisania, czyli sekwencję :=. Zawsze, kiedy będziesz chciał nadać wartość jakiejś zmiennej będziesz musiał skorzystać właśnie z tego operatora. Wykorzystanie zmiennej w instrukcji Writeln wygląda identycznie jak w przypadku stałych. Kompilator po napotkaniu sekwencji x*x jest w stanie przetłumaczyć ją na kod maszynowy, bowiem „wie" czym jest x i jaka jest pod tą zmienną wartość.

W naszym przykładzie wykorzystaliśmy tylko jedną zmienną. Może ich być jednak znacznie więcej. Przy czym, podobnie jak w przypadku deklaracji stałych, słowo kluczowe var używamy tylko raz - na początku. Każdą deklarację zmiennej umieszczamy w oddzielnej linii i kończymy średnikiem. Gdybyśmy deklarowali kilka zmiennych tego samego typu (np. Integer) moglibyśmy wymienić je po przecinku. Spójrz na przykładowy program:

```
program PO17;
var x,y : Integer;
      A : string;
begin
   X:=10;
   Y:=(x*2)+1;
   A:='Karol';
   Writeln(A,' ma ',Y,' lat.');
end.
```

Wynik:

```
Karol ma 21 lat.
```

W powyższym przykładzie stworzyliśmy trzy zmienne. Dwie (x i y) typu Integer - czyli liczby całkowite i jedną o nazwie A typu String - co oznacza napis. Następnie w programie przypisaliśmy zmiennej x wartość 10. W kolejnej linii przypisaliśmy zmiennej Y wartość wyrażenia (x*2)+1. Zanim więc przypisana zostanie wartość do zmiennej Y musi być uprzednio obliczona wartość tego wyrażenia. Ponieważ zmienna X ma wartość 10, zmienna Y przyjmie wartość

(10*2)+1, czyli 21. W kolejnej linii instrukcją Writeln wyświetlamy, w znany już sposób, komunikat.

Zwróć uwagę, że zanim danej zmiennej przypisana zostanie wartość, obliczone zostanie wyrażenie umieszczone po prawej stronie. Taka jest właściwość operatora :=. Stąd po prawej stronie tej instrukcji moglibyśmy użyć także zmiennej X. Spójrz na fragment programu:

```
X:=2;
X:=X*X;
```

Nadaliśmy zmiennej X wartość 2. Następnie przypisaliśmy jej wartość wyrażenia X*X. Ponieważ w tym momencie X wynosił 2, wyrażenie X*X jest równoważne 2*2, czyli 4. Taki też wynik został przypisany zmiennej X. Oto podobny zapis:

```
X:=1;
X:=X+1;
```

Zmiennej X przypisujemy wartość 1. W drugiej linijce przypisujemy tej samej zmiennej wartość X+1. Ponieważ X w tym momencie wynosiło 1, więc nowa wartość będzie 1+1, czyli 2.

4.3. Typy proste

Mógłbyś zadać pytanie - po co jest tyle typów danych? Cóż to za różnica, czy coś jest liczbą czy napisem. Otóż jest to różnica bardzo znacząca. Pamiętaj, że kiedy program jest tłumaczony na kod maszynowy, procesor nie może mieć żadnych wątpliwości, co do tego, co i jak ma wykonać. Musi mieć wszystko jasno sprecyzowane. Gdybyśmy nie określili, że np. zmienna X jest typu Integer (liczba całkowita), to co miałby zrobić procesor, a kompilator jak miałby przetłumaczyć sekwencję x: =x*x, gdy zmienna X zawierałaby napis „Karol"? Co oznaczałoby, że X: =" Karol" * " Karol" ? Jak to wykonać? Nie da się. Bowiem nie można mnożyć napisów.

Jeśli na początku programu zaznaczymy, że X jest typu Integer, to gdy kompilator napotka w programie zapis x: = 'Karol' ; wyświetli komunikat o błędzie, w którym nas poinformuje, że próbujemy przypisać zmiennej określonego typu wartość innego typu. Przekonaj się o tym samodzielnie. Wpisz taki program:

```
program PO18;
var X : Integer;
begin
   X:='Karol';     {tu jest błąd}
   X:=X*X;
   Writeln(X);
end.
```

Spróbuj go skompilować wciskając [F9].

Jak widzisz typy danych są ważne i niezbędne dlatego, że kompilator lub procesor mogłyby nie wiedzieć jak wykonać niektóre instrukcje. Ponieważ zmienne mają swój typ, konkretnie zdefiniowany jest zbiór wszystkich operatorów dla nich, czyli czynności jakie są dozwolone. Jeśli bowiem mamy zmienną X typu Integer, cóż miałoby jej zostać przypisane z wyrażenia: X:=2/3? Takie działanie jest niedopuszczalne dla liczb całkowitych, ale dla liczb rzeczywistych (typ Real) oczywiście tak.

Druga, ważna cecha, która odróżnia zmienne od stałych i wymaga podziału tych pierwszych na typy to fakt, że zmienne zajmują pewną, określoną ilość pamięci. Każdy komputer posiada tzw. pamięć RAM. Jest to kilka, czy kilkadziesiąt MB. W programach pisanych w Turbo Pascalu „widzimy" tylko kilkadziesiąt KB z tej całej objętości. Kiedy deklarujemy (tworzymy) zmienną, rezerwujemy określoną ilość bajtów na jej potrzeby. Dzięki temu, gdy przypisujemy do zmiennej pewną wartość, zapamiętywana jest ona w tym, przydzielonym uprzednio miejscu. Ponieważ ilość pamięci jest ograniczona - ograniczona jest również ilość zmiennych, jakie możemy utworzyć. Aby gospodarować tym jak najlepiej mamy do dyspozycji sporą ilość typów. Zmienne jednego rodzaju zajmują 1 bajt, innego 2, jeszcze innego aż 100! Aby zapamiętać małą liczbę całkowitą wystarczy 1 bajt. Dla liczb rzeczywistych potrzeba już więcej. Dla napisów trzeba tyle bajtów, ile liter będzie - w najgorszym przypadku - zawierała nasza zmienna.

Oto lista najczęściej wykorzystywanych typów:

- Byte - liczby całkowite od 0 do 255
- Word - liczby całkowite od 0 do 65.535
- Integer - liczby całkowite od -32768 do 32767
- Longint - liczby całkowite od -2147483648 do 2147483647
- Real - liczby rzeczywiste od $-2.9*10^{-39}$ do $1.7*10^{38}$
- Char - znak **ASCII**
- String - ciąg znaków **ASCII**
- Boolean - **typ** logiczny: TRUE **lub** FALSE

4.3.1. Typ całkowite

Typy całkowite to: Byte, Word, Integer, Longint. Każda zmienna określona w ten sposób może przechowywać liczby całkowite z określonych (przedstawionych wcześniej) przedziałów. Dozwolone są wszystkie podstawowe operatory arytmetyczne, a więc:

- + - dodawanie
- - - odejmowanie
- * - mnożenie

Operator dzielenia / jest zabroniony, gdyż wynik dzielenia może nie być liczbą całkowitą. Jednak w zamian za to dostępne są dwa inne operatory:

- Div - dzielenie całkowite
- Mod - reszta z dzielenia całkowitego

Dzielenie całkowite, to wynik tradycyjnego dzielenia bez części ułamkowej. Nie wolno mylić tego z zaokrąglaniem. Natomiast reszta z dzielenia, to właśnie ta część, która pozostaje przy dzieleniu całkowitym. Tutaj również nie można tego utożsamiać z częścią występującą po przecinku w zapisie dziesiętnym. Przykładowo wyrażenie 3 div 2 jest równe 1, bowiem tylko jednak dwójka mieści się w całości w trójce. Jednak 3 mod 2 nie wynosi 5, jak ktoś mógłby pomyśleć (3/2=1.5), tylko 1. Gdyż w naszej trójce mieści się jedna dwójka i pozostaje jeszcze reszta 1 (3-2=1).

4.3.2. Typ rzeczywiste

Zmienne typu Real to liczby rzeczywiste, tzn. wszystkie mieszczące się w granicach tego typu. Dopuszczalne dla nich są wszystkie podstawowe operatory, a więc dodawanie (+), odejmowanie (-), mnożenie (*) i dzielenie (/). Nie wolno jednak stosować operatorów dzielenia specyficznych dla liczb całkowitych: div i mod.

Większość funkcji matematycznych dostępnych w Turbo Pascalu operuje właśnie na tym typie. Przykładowe funkcje, z których możesz skorzystać to:

- Sin - oblicza wartość funkcji sinus
- Cos - oblicza wartość funkcji cosinus
- Tan - oblicza wartość funkcji tangens
- Sqrt - oblicza pierwiastek kwadratowy
- Exp - podnosi liczbę do potęgi e

Liczby typu Real wyświetlane za pomocą instrukcji Write lub Writeln wykorzystują notację naukową (z E). Spójrz na przykład:

```
program P019;
var x : Real;
begin
   X:=1/3;
   Writeln(x);
end.
```

Wynik:

```
3.3333333333E-01
```

Nie jest to sposób przyjazny użytkownikowi. Na szczęście istnieje metoda naturalnego zapisu liczb. Wpisz poniższy program:

```
program PO 2 0;
var x : Real;
begin
   X:=i/3;
   Writeln(x:0:3);
end.
```

Wynik:

```
0.333
```

Tym razem wynik wyświetlony jest poprawnie. W instrukcji Writeln dopisaliśmy sekwencję :0:3. Druga cyfra (3) określa dokładność, z jaką ma być wyświetlona liczba. W naszym przykładzie jest to liczba z dokładnością do trzech miejsc po przecinku. Pierwsza cyfra określa na ilu znakach ma być dana liczba zapisana. Cyfra 0 oznacza, że ta część ma nie być uwzględniana. Dokładnie tym się zajmiemy i powrócimy do tego zapisu, kiedy omówimy instrukcje iteracyjne.

4.3.3. Typ Boolean

Typ Boolean to tzw. typ logiczny. Zmienne tego typu mogą przyjmować tylko dwie różne wartości. TRUE - co oznacza „prawda" i FALSE, czyli „fałsz". Nie dopuszczalne są tutaj żadne operatory arytmetyczne, bo cóż miałoby znaczyć „prawda plus fałsz"? Dostępne są jednak operatory logiczne.

- = - operator równoważności
- >-operator „większe"

- >= - operator „większe bądź równe"
- < - operator „mniejsze"
- <= - operator „mniejsze bądź równe"
- <> - operator „różne"

Możemy przykładowo napisać:

```
program PO21;
var A : Boolean;
begin
    A:=2=3;
    Writeln(A)
end.
```

Wynik:

```
FALSE
```

Jak mogliśmy się przekonać zmienna A przyjęła wartość FALSE, czyli fałsz. Bowiem przypisaliśmy jej wartość wyrażenia „2=3". Operator równoważności „=" porównuje ze sobą dwie liczby (lub ciągi znaków) i jeśli są sobie równe, zwraca wynik TRUE, a w przeciwnym wypadku właśnie FALSE. Oto inny przykład:

```
program P022;
var A,B,C : Boolean;
begin
   A:=3>=3;
   B:=4<>4;
   C:='Karol'='karol';
   Writeln(A);
   Writeln(B);
   Writeln(C);
end.
```

Wynik:

```
TRUE
FALSE
FALSE
```

Jak pokazują wyniki, wartość wyrażenia „3>=3" wynosi TRUE - czyli „prawda", **bowiem** rzeczywiście trzy jest większe bądź równe trzem. Wyrażenie „4<>4" ma wartość FALSE. Bowiem nie jest prawdą, że cztery jest różne od czterech. Podobnie zmienna C ma wartość FALSE, gdyż napis „Karol" nie jest równy napisowi „karol" (istotna jest wielkość znaków).

Takie wyrażenia logiczne można umieszczać w nawiasach i w ten sposób tworzyć bardziej rozbudowane wyrażenia. Możemy wówczas korzystać z trzech dodatkowych operatorów:

- AND - iloczyn logiczny
- OR - suma logiczna
- NOT - negacja logiczna

Jeśli mamy dwa wyrażenia połączone iloczynem logicznym, to całe wyrażenie przyjmie wartość TRUE wówczas, kiedy oba z podwyrażeń są również równe TRUE. Jeśli chociaż jedno z podwyrażeń wynosi FALSE, całe wyrażenie także przyjmie tę wartość. Gdy wyrażenia połączone są sumą logiczną - całe wyrażenie przyjmie wartość TRUE, jeśli chociaż jedno (lub oba) z podwyrażeń przyjmie tę wartość. Negacja logiczna zmienia wartość wyrażenia na przeciwną, a więc z TRUE na FALSE i z FALSE na TRUE. Poniżej zamieściłem tabele prawd, które mogą się przydać przy okazji tworzenia instrukcji warunkowych.

A	B	A **AND** B
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

A	B	A OR B
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

A	NOT B
TRUE	FALSE
FALSE	TRUE

Rysunek 5.1. Tabele logiczne

Oto przykład programu demonstrującego nowe operatory logiczne:

```
program PO23;
var A,B : Boolean;
begin
   A:=TRUE;
   A:=not A ;
   Writeln(A);
   B:=(not  (4=3))  and  (4=4);
   Writeln(B);
end.
```

Wynik:

```
FALSE
TRUE
```

Przeanalizujmy program. Zmiennej A przypisano na początku wartość TRUE. Następnie tej samej zmiennej przypisaliśmy zanegowaną wartość zmiennej A. Ponieważ była to wartość TRUE, po negacji wyniosła FALSE i taką wartość na końcu zmienna A zawierała. Taką też wartość wyświetliła pierwsza instrukcja Writeln. Zmiennej B przypisano wynik bardzo rozbudowanego wyrażenia. Składa się ono z dwóch podwyrażeń. Pierwsze z nich to not (A=3) i drugie 4 = 4. Połączone są one spójnikiem logicznym AND (iloczyn logiczny), a wiec wynik całego wyrażenia wyniesie TRUE jeśli oba te mniejsze wyrażenia są równe TRUE. Jeśli chociaż jedno jest fałszywe - całe wyrażenie także będzie fałszywe. Drugie podwyrażenie jest oczywiste. 4=4 to bez wątpienia TRUE (prawda). Pierwsze wyrażenie wymaga wyjaśnienia. Wykonywane jest w pierwszej kolejności działanie 4=3 i jego wynik wyniesie oczywiście FALSE. Następnie wynik ten zostanie zanegowany, gdyż poprzedzony jest słowem not. Negacja fałszu jest prawdą. W ten sposób oba podwyrażenia są prawdziwe, a więc i całe wyrażenie również. Taką też wartość wyświetliła druga instrukcja Writeln.

4.3.4. Typ Char

Każda litera dostępna na klawiaturze (i jeszcze kilka więcej) posiada swój unikalny numer w tzw. tabeli kodów ASCII. Przykładowo mała literka „a" ma swój kod 97. Wielka litera „A" kod 65. Kod klawisza [ENTER] wynosi 13, a spacji 32.

Dodatek C zawiera tabelę kodów ASCII, które są kodami innych znaków klawiatury.

Zmienne typu Char mogą przetrzymywać takie pojedyncze litery. Jeśli chcemy imiennej przypisać jakiś znak, umieszczamy go w apostrofach. Możemy również posłużyć się jego kodem ASCII (szczególnie pożyteczne, gdy na klawiaturze nie da się wstawić określonego znaku - np. [ENTER]) poprzedzonym znakiem „#".

Oto przykład:

```
program P024;
var Znak : Char;
begin
   Znak:='A';
   Writeln(Znak);
   Znak:=#97;
   Writeln(Znak);
end.
```

Wynik:

```
A
a
```

Korzystając ze znaku # i rozszerzonych znaków ASCII można tworzyć proste rysunki, np. tabele, czy ramki.

4.3.5. Typ String

Zmienne typu Char pozwalały na przechowanie tylko pojedynczych znaków. Typ String jest ich pewnym rozszerzeniem. To jakby określona ilość zmiennych typu Char ułożonych jedna za drugą i obsługiwana przez niektóre instrukcje jako całość. Jest to więc typ „napisowy". Spójrz na przykład:

```
program P025;
var Napis : String;
begin
   Napis:='Turbo  Pascal';
   Writeln(Napis);
end.
```

W powyższym programie do zmiennej Napis przypisaliśmy tekst „Turbo Pascal". Umieściliśmy go, podobnie jak znaki, w apostrofach.

Na zmiennych napisowych nie można wykonywać żadnych operacji arytmetycznych. Nie można ich dodawać, odejmować, mnożyć ani dzielić. Takie operacje nie miałyby sensu. Możliwe jest tylko łączenie napisów i dokonuje się tego operatorem łączenia, który „wygląda" identycznie jak operator dodawania. Jest to znak plus „+".

Typ String jest typem specyficznym, bowiem przy deklaracji można określić wielkość napisu, jaki może pomieścić dana zmienna. W naszym poprzednim przykładzie tego nie określiliśmy. Kompilator wykorzystał domyślną wartość równą 255. Możemy jednak jawnie ją określić. Spójrz na nasz kolejny program:

```
program P026;
var Napis : String[20];
begin
   Napis:='Turbo ';
   Napis:=Napis+'Pascal';
   Writeln(Napis);
end.
```

W programie utworzyliśmy zmienną tekstową o nazwie Napis. Jawnie określiliśmy jej wielkość na 20 znaków (wielkość podajemy w nawiasach kwadratowych). Oznacza to, że zmienna Napis jest jakby połączeniem 20 zmiennych typu Char. Nie można więc pomieścić w niej więcej niż 20 liter. Mniej, oczywiście tak. Aby pokazać sposób łączenia napisów skleiliśmy ze sobą wyrazy „Turbo " i „Pascal" tworząc w ten sposób jeden napis „Turbo Pascal".

Pascal udostępnia nam funkcję Length, która zwraca długość napisu, przekazanego jako argument. Oto prosty przykład:

```
program P027;
var A : String[10];
begin
   A:='Kamil';
   Writeln(Length(A));
end.
```

Wynik:

```
5
```

Długość napisu nie musi być równa jego maksymalnej długości, jak pokazuje powyższy przykład.

Ponieważ napisy są połączeniem wielu zmiennych typu Char, przypisując zmiennej tekstowej jakiś ciąg, w efekcie każdej kolejnej zmiennej znakowej wchodzącej w skład napisu przypisywana jest kolejna litera z danego ciągu. Programista ma możliwość odwoływania się do poszczególnych liter dzięki nawiasom kwadratowym. Spójrz na przykład:

```
program P028;
var N : String[30];
begin
   N:='Pascal';
   Writeln(N[1]);
   Writeln(N[2]);
   Writeln(N[3]);
   Writeln(N[4]);
   Writeln(N[5]);
   Writeln(N[6]);
   N[1]:='p';
   Writeln(N);
end.
```

Wynik:

```
p
a
s
c
a
l
pascal
```

W przedstawionym powyżej przykładzie zadeklarowaliśmy jedną zmienną napisową. Może ona zawierać tekst o długości nie przekraczającej 30 znaków, gdyż kompilator tylko tyle pamięci dla niej zarezerwował. Następnie w programie głównym przypisaliśmy zmiennej treść „Pascal". Korzystając z nawiasów kwadratowych i instrukcji Writeln wyświetliliśmy kolejne litery tekstu. Odwołujemy się do nich podając numer litery w nawiasach kwadratowych. Następnie zmieniliśmy pierwszą literę na małe „p" i wyświetliliśmy napis w całości.

Odwoływaliśmy się do poszczególnych liter poprzez ich numer przekazany jako argument. Numeracja rozpoczyna się od 1 - dla pierwszej litery, 2 - dla drugiej, itd. A co jest zapisane pod indeksem 0? Długość napisu! Kiedy korzystasz z funkcji Length, tak naprawdę odczytujesz zerowy indeks zmiennej. Kiedy A: = 'Kamil', toA[0] = length(A) = 5.

Turbo Pascal dostarcza programiście trzy ciekawe instrukcje operujące na zmiennych napisowych. Są to:

- Copy - służy do kopiowania fragmentów tekstu zmiennych napisowych
- Delete - umożliwia wycinanie fragmentów tekstu
- Pos - pozwala na odszukiwanie fragmentów tekstu

Funkcja Copy pozwala na skopiowanie z danego tekstu pewnego fragmentu. Oczekuje trzech argumentów. Pierwszym z nich jest zmienna napisowa, z której chcemy coś skopiować. Drugi argument to numer litery, od której chcemy rozpocząć kopiowanie, a trzeci argument to ilość kopiowanych liter. Oto fragment programu:

```
var A,B : String;

A:='Jan Nowak urodzony w Gdańsku';
B:=Copy(A,4,5);
Writeln(B);
```

Przypisaliśmy zmiennej A pewien napis, w którym znajduje się również nazwisko "Nowak". Zaczyna się ono od 4 znaku i składa z 5 liter. Dzięki funkcji Copy kopiujemy ze zmiennej A i zapamiętujemy w zmiennej B fragment rozpoczynający się 4 znakiem i mający właśnie długość 5 liter. W ten sposób instrukcja Writeln wyświetli samo nazwisko „Nowak".

Natomiast procedura Delete usuwa pewien fragment tekstu z napisu. Ponownie przeanalizuj poniższy fragment:

```
var A : String;

A:='Jan Nowak urodzony w Gdańsku';
Delete(A,4,6);
Writeln(A);
```

Tym razem wyświetlony zostanie komunikat „Jan urodzony w Gdańsku". Usunięte więc zostało nazwisko i spacja po nim. Instrukcja Delete również oczekuje trzech argumentów, analogicznych do funkcji Copy, a więc zmiennej napisowej, numeru litery, od której chcemy usunąć fragment tekstu i ilości znaków, jakie chcemy usunąć.

Funkcja Pos pozwala na odszukanie w jednym tekście innego napisu. Oczekuje dwóch argumentów. Pierwszym jest szukany wzorzec, zaś drugim zmienna napisowa, którą przeszukujemy. Jeśli podtekst zostanie odnaleziony, funkcja zwróci numer litery, od której się on rozpoczyna. Jeśli nie zostanie odnaleziony, funkcja Pos zwróci wartość 0. Oto fragment programu, który rozwieje wszelkie wątpliwości:

```
var A : String;

A:='Jan Nowak urodzony w Gdańsku';
Writeln('Pozycja wystąpienia nazwiska Nowak:  ',<
         Pos('Nowak',A));
```

4.4. Wprowadzanie danych do programu

Znamy już podstawowe informacje na temat budowy programu w Turbo Pascalu. Potrafimy tworzyć własne stałe, zmienne, nadawać im wartości, dokonywać różnych prostych (lecz nie tylko) obliczeń, wyświetlać wyniki swoich działań, itd. Jednak nasze programy posiadały dotychczas jedną zasadniczą wadę. Wykonywały

zadanie z góry przez nas (programistę) określone. Nie operowały na informacjach, które chciałby przekazać im użytkownik. Przykładowo spójrz na poniższy program:

```
program P029;
var x : Integer;
begin
   x:=30;
   Writeln('sin(',x, ') = ',sin(x*pi/180) :0:3) ;
end.
```

Wynik:

```
sin(30.000)=0.500
```

Powyższy program oblicza wartość funkcji sinus dla 30 stopni. Jednak tylko dla 30 stopni! Taki program jest praktycznie bezużyteczny. Znacznie lepiej by było, gdyby użytkownik został poproszony o wprowadzenie kąta, którego sinus chce obliczyć. I właśnie tego nauczymy się w tym momencie.

Funkcje trygonometryczne dostępne w Turbo Pascalu, a więc sin, cos i tan operują na radianach. Ponieważ Czytelnik przyzwyczajony jest zapewne do skali stopniowej - aby możliwe były takie obliczenia - wystarczy zamienić przekazane przez użytkownika stopnie (np. 30) na radiany. W powyższym programie dokonaliśmy tego zgodnie ze wzorem Radiany:=Stopnie*PI/180.

Procedury Write i Writeln „wysyłały" napisy (lub ogólnie: przekazane argumenty) na tzw. standardowe wyjście. Jest nim ekran monitora. Procedury Read i ReadLn w bardzo podobny sposób „pobierają" napisy (lub dane innego typu) ze standardowego wejścia. Tym z kolei jest klawiatura.

Zacznijmy od przykładu.

```
program P030;
var X : Integer;
begin
   Write('Podaj kąt >');
   Read(x);
   Writeln('sin(',x, ') = ',sin(x*pi/180) :0:3) ;
end.
```

Wynik:

```
Podaj kąt > 30
sin(30)=0.500
```

Skompiluj i uruchom powyższy program. Wyświetlony zostanie komunikat *„Podaj kąt >"* i obok będzie migał kursor. Kiedy wpiszesz z klawiatury jakąś liczbę całkowitą i wciśniesz klawisz [ENTER] wyświetli się wartość sinusa podanego przez Ciebie kąta.

Przyjrzyjmy się źródłu. Najpierw wyświetliliśmy komunikat, który informuje użytkownika co ma uczynić. Wykorzystaliśmy w tym celu procedurę Write. Dlaczego nie Writeln? Bo chcieliśmy, aby wprowadzane liczby pojawiały się obok naszego napisu (czyli tam, gdzie pozostał kursor). Gdybyśmy użyli Writeln wpisywana przez nas liczba pojawiałaby się na dole. Sprawdź sam! W kolejnej linii programu wywołujemy instrukcję Read. To właśnie ona „czyta" z klawiatury to, co my wpisujemy. Zapamiętuje to w zmiennej przekazanej jej jako argument. Przed tym jednak - automatycznie - konwertuje wpisane przez nas dane do odpowiedniego typu. W naszym przykładzie do typu Integer, bowiem zmienna X tak właśnie została zadeklarowana. Gdybyśmy zadeklarowali ją jako Real (sprawdź samodzielnie) konwertowałaby do tego typu. Jest to bardzo ważne spostrzeżenie. Bowiem teraz, jeśli przy wprowadzaniu liczby z klawiatury użyjesz jakiejś litery, czy też znaku kropki (jak dla liczby rzeczywistej) wystąpi błąd. W następnej linii - znaną już instrukcją Writeln - wyświetlamy wynik działania.

Sposób wykorzystania procedury ReadLn jest analogiczny do przedstawionej przed chwilą. Omówmy jednak teraz różnice między nimi.

Instrukcja Read czyta wprowadzone przez użytkownika znaki aż do napotkania spacji lub klawisza enter. Nadaje się wiec do pobierania pojedynczych wyrazów, liczb, etc. Nie pozwala natomiast na wprowadzenie ciągu, w którym znajdują się odstępy, np. imienia i nazwiska. Wówczas dobrze jest zastosować ReadLn. Bowiem ta instrukcja pobiera całe linie tekstu, a więc aż do napotkania klawisza [ENTER]. ReadLn możemy również wykorzystać bez podawania żadnego argumentu. Wówczas wprowadzone przez nas dane nie będą po prostu nigdzie zapamiętywane. Jaki jest tego cel? Na przykład zatrzymanie programu. Zwróć uwagę, że wszystkie dotychczas napisane przez nas programy bardzo szybko się wyłączały i aby zobaczyć efekt ich działania musieliśmy korzystać z klawiszy [ALT+F5]. Znając procedurę ReadLn możemy użyć jej (bez żadnych argumentów) na końcu każdego z naszych programów. Wówczas nie będą się one natychmiast wyłączać, a dopiero wówczas, kiedy my tego zechcemy wciskając klawisz [ENTER].

Spójrz na nasz kolejny program:

```
program PO 31;
var Dane : String;
    Wiek : Byte;
begin
   Write('Jak się nazywasz? >');
   ReadLn(Dane);
   Write('Ile masz lat? >').;
   Read(Wiek);
   Writeln('Jesteś ',Dane,' i masz ',Wiek,' lat.');
   ReadLn;
end.
```

Wynik:

```
Jak się nazywasz? >Jan Nowak
Ilę masz lat? >21
Jesteś Jan Nowak i masz 21 lat.
```

Przeanalizuj również następny program:

```
program  P032;
var Imie : String;
begin
   Write('Podaj swoje imię >');
   ReadLn(Imie);
   Writeln('Czy Twoje imię jest ładne? ',
            Imie='Karol');
   ReadLn;
end.
```

Spójrz na przykładowe wyniki:

```
Podaj swoje imię > Karol
Czy Twoje imię jest ładne? TRUE

Podaj swoje imię > Piotr
Czy Twoje imię jest ładne? FALSE
```

Jak widzisz, użytkownik proszony jest o wprowadzenie swojego imienia. Następnie wyświetlany jest komunikat i wartość wyrażenia `Imie='Karol'`. Co sprawia, że jeśli użytkownik wprowadzi takie właśnie imię, uznane ono zostanie za ładne (wyrażenie przyjmie wartość TRUE), a jeśli jakiekolwiek inne imię - nie będzie ono ładne (FALSE).

Widzimy tu pewien problem. My, jako programiści, znamy słowa TRUE i FALSE, i potrafimy je interpretować. Jednak znacznie lepiej by było, gdyby komunikat był bardziej rzeczowy i opisowy, aby każdy był w stanie go zrozumieć. Nauczymy się tego w następnym rozdziale.

5. Instrukcje warunkowe

Nauczyliśmy się już tworzyć wyrażenia logiczne, a więc takie które przyjmowały wartości TRUE lub FALSE. Dzięki temu potrafiliśmy tworzyć pewne warunki. Mogliśmy, np. sprawdzić czy dana liczba jest parzysta, czy też nie. Czy wprowadzone imię, jest ładne, czy też brzydkie (według nas). Czy wreszcie użytkownik ma określoną ilość lat, pozwalającą mu na uruchomienie programu. Jednak takie warunki na niewiele nam się zdały, gdyż jedyne co mogliśmy z nimi zrobić, to wyświetlić ich wartość w postaci wyrazów TRUE lub FALSE instrukcjami Write i Writeln. W tym rozdziale nauczymy się dokładniej reagować na pewne warunki, a więc w przypadku ich spełnienia (TRUE) wykonywać określone instrukcje.

5.1. Instrukcja IF

Instrukcja if pozwala na zadawanie pytań *„Jeśli"*. Oto jej składnia:

```
if warunek then instrukcja [else instrukcja];
```

Powyższą linijkę należy rozumieć w następujący sposób. Pomiędzy słowami kluczowe if i then umieszczamy instrukcję warunkową, a więc taką, której wynik przyjmuje wartość typu Boolean (TRUE lub FALSE). Kiedy warunek zostanie spełniony (ma wartość TRUE) wykonywana jest instrukcja znajdująca się po słowie then. Jeśli warunek nie jest spełniony, wykonuje się instrukcja po słowie kluczowym else (oznaczającym „w przeciwnym wypadku"). Tę część instrukcji warunkowej możemy pominąć i wówczas, kiedy warunek przyjmie wartość FALSE nie wykona się żadna instrukcja. Oto obiecany w poprzednim rozdziale przykład:

```
program PO33;
var Imie : String;
begin
   Write('Podaj swoje imię >');
   ReadLn(Imię);
   if imie='Karol' then Writeln('Masz ładne imię')
       else Writeln('Masz brzydkie imię');
   ReadLn;
end.
```

Przykładowe wyniki programu:

```
Podaj swoje imię > Karol
Masz ładne imię
Podaj swoje imię > Piotr
Masz brzydkie imię
```

Program działa więc w ten sposób, że w zależności od tego, czy zmienna Imie zawiera napis „Karol", czy też nie - wyświetla stosowny komunikat.

Wykorzystując instrukcję warunkową, można tworzyć znacznie ciekawsze programy. Spójrz na nasz kolejny przykład:

```
program P034;
var Liczba : Integer;
begin
   Write('Podaj liczbę całkowitą >');
   ReadLn(Liczba);
   if Liczba mod 2-0 then
        Writeln('To jest liczba parzysta')
   else Writeln('To nie jest liczba parzysta');
   ReadLn;
end.
```

Powyższy program sprawdza, czy wprowadzona liczba jest parzysta, czy też nie. Wykorzystuje w tym celu operator mod, zwracający resztę z dzielenia. Oto prosty program dzielący liczby:

```
program P035;
var A,B : Real;
begin
   Write ('Podaj dzielną >'•)";
   ReadLn(A);
   Write('Podaj dzielnik >');
   ReadLn(B);
   if B=0 then
        Writeln('Nie można dzielić przez zero!')
   else Writeln(A:0:3,'/',B:0:3,'=',A/B:0:3);
   ReadLn;
end.
```

Przykładowe wyniki programu:

```
Podaj dzielną > 10
Podaj dzielnik > 2
10.000/2.000=5.000
```

```
Podaj dzielną > 15
Podaj dzielnik > 0
Nie można dzielić przez zero
```

Składnia instrukcji if pozwala na użycie tylko jednej instrukcji po słowie kluczowym then i jednej po słowie kluczowym else. Cóż mamy uczynić, jeśli chcemy wykonać więcej operacji, gdy dany warunek zostanie spełniony? Wystarczy skorzystać z tzw. instrukcji złożonej. Instrukcja złożona to blok instrukcji objęty słowami kluczowymi begin i end. Kompilator taki blok „widzi" jak pojedynczą instrukcję. Spójrz na przykład:

```
program P036;
var Wiek : Integer;
begin
   Write('Ilę masz lat? ');
   ReadLn(Wiek);
   if (Wiek>=18) and (Wiek<=20) then
   begin
       Writeln('Twój wiek odpowiada wymaganiom <
                stawianym przez program.');
     Writeln('To już druga instrukcja wewnątrz <
              warunku.');
   end;
   ReadLn;
end.
```

Zwróć również uwagę na nasz warunek. Ma on formę rozbudowaną. Składa się z dwóch warunków cząstkowych połączonych spójnikiem and.

Ponieważ chcieliśmy wykonać więcej niż jedną instrukcję, gdy nasz warunek zostanie spełniony, skorzystaliśmy w powyższym przykładzie z instrukcji złożonej, a więc po słowie kluczowym then umieściliśmy blok begin. . . end, w którym zawarliśmy poszczególne instrukcje. Zwróć uwagę, że po słowie kluczowym end kończącym instrukcję jest średnik. Został on umieszczony, ponieważ składnia instrukcji if wymaga po słowie then jednej instrukcji zakończonej tym znakiem. Nasz blok begin. . . end „widziany" jest właśnie jako ta jedna instrukcja. Spójrz na szablon poniżej.

```
if warunek then
begin

end else
begin

end;
```

To przykładowy szablon rozbudowanej instrukcji warunkowej. Zarówno sekwencja po słowie then jak i po słowie else są instrukcjami złożonymi. Jednak (jak zapewne pamiętasz) po słowie end (przed else) nie ma średnika, bowiem i w zwykłej instrukcji go tutaj nie było.

Instrukcje warunkowe i instrukcje złożone można dowolnie zagnieżdżać. Spójrz na kolejny szablon, który to pokazuje:

```
if warunek then
begin
   if warunek then
      if warunek then
      begin
         .
      end;
end;
```

Oto kolejny program demonstrujący wykorzystanie instrukcji warunkowych:

```
program P037;
var Napis,Szukaj : String;
    Pozycja : Integer;
begin
   Write('Podaj dłuższy napis: ');
   ReadLn(Napis);
   Write(' Podaj szukany tekst: ');
   ReadLn(Szukaj);
   Pozycja:=Pos(Szukaj,Napis);
   if Pozycja=0 then Writeln('Nie znaleziono <
                          szukanego tekstu.')
                else Writeln ('Znaleziono szukany <
                       tekst na pozycji ',Pozycja);
   ReadLn;
end.
```

Spróbuj teraz napisać kilka prostych programów wykorzystujących instrukcje warunkowe. Jest to bowiem zagadnienie bardzo ważne, gdyż będziesz się z nim stykał w niemalże każdym swoim programie.

5.2. Instrukcja CASE

Kolejną instrukcją warunkową w języku Turbo Pascal jest case. Dzięki niej możemy w prosty sposób stworzyć listę warunków i odpowiadających im instrukcji.

Składnia jest następująca:

```
case wyrażenie of
Wartość  : instrukcja1;
Wartość2 : instrukcja2;

[else instrukcja;]
end;
```

Wyrażenie musi przyjmować wartość porządkową, a więc znak, bądź liczbę całkowitą. Wykonanie instrukcji case przebiega w ten sposób, że obliczana jest wartość danego wyrażenia i porównywana z wartościami umieszczonymi na liście. Jeśli któraś z wartości na liście jest jej równa, wykonana zostanie określona instrukcja i zakończy się blok case..end. Jeśli umieścimy opcjonalną sekwencję else i nie zostanie odnaleziona wartość wyrażenia na liście, wykona się instrukcja po tym słowie. Spójrz na przykład:

```
program P038;
var Znak : Char;
     A,B : Real;
Begin
    Writeln('Prosty kalkulator.');
    Writeln(' + - dodawanie');
    Writeln(' * - mnożenie');
    Write('Wybierz:');
    Read(Znak);
    Writeln;
    case Znak of
     '+': begin
            Write('Podaj liczbę A>'); ReadLn(A);
            Write('Podaj liczbę B>'); ReadLn(B);
            Writeln('Wynik:',A:0:3,'+',B:0:3,'=',
                     A+B:0:3);
          end;
```

```
      '*': begin
             Write('Podaj liczbę A>'); ReadLn(A);
             Write('Podaj liczbę B>'); ReadLn(B) ;
             Writeln('Wynik: ',A:0:3,'*',B:0:3,'=',
                     A*B:0:3);
           end;
       else Writeln('Nieznane działanie');
    end;
    ReadLn;
end.
```

Wynik:

```
Prosty kalkulator
+ - dodawanie
* - mnożenie
Wybierz: +

Podaj liczbę A>2
Podaj liczbę B>4
Wynik: 2.000+4.000=6.000
```

W powyższym przykładzie dla poszczególnych przypadków w instrukcji case użyliśmy instrukcji złożonych, aby móc umieścić więcej mniejszych instrukcji.

Spróbuj teraz samodzielnie rozbudować powyższy program tak, aby obsługiwał jeszcze działania: odejmowanie i dzielenie. Zwróć uwagę, aby w przypadku dzielenia nie dopuszczał do zastosowania 0, jako dzielnika.

Z instrukcji case korzysta się często wówczas, kiedy w programie tworzymy pewnego rodzaju menu (jak w naszym przykładzie). Jednak znajdzie ona zastosowanie jeszcze w innych miejscach znacznie upraszczając zapis danego programu.

6. Instrukcje iteracyjne

Instrukcje iteracyjne umożliwiają wykonanie pewnego bloku instrukcji określoną ilość razy, np. 10 razy. Pozwalają także na wykonywanie danego bloku tak długo, aż pewien warunek zostanie spełniony lub też nie. Korzystać z nich będziemy często, chcąc „zapętlić" dany fragment programu, a więc by wykonał się on więcej razy. Turbo Pascal udostępnia trzy rodzaje instrukcji iteracyjnych: for, while i repeat.

6.1. Instrukcja FOR

Składnia instrukcji jest następująca:

```
for licznik:=start to stop do instrukcja;
```

Zmiennej licznik przypisana jest wartość start, a następnie wykonuje się instrukcja umieszczona po słowie do. Kiedy zostanie ona wykonana, wartość zmiennej licznik zwiększana jest o jeden i ponownie wykonuje się instrukcja. Trwa to tak długo, dopóki zmienna licznik zawiera wartość mniejszą bądź równą wartości stop.

Spójrz na przykład:

```
program P039;
var i : Integer;
begin
   for i:=1 to 5 do Writeln('Linia nr. ',i);
   ReadLn;
end.
```

Wynik:

```
linia nr. 1
linia nr. 2
linia nr. 3
linia nr. 4
linia nr. 5
```

Jak widzisz, aby skorzystać z tej instrukcji potrzebujemy pewnej zmiennej - licznika. Musi on być liczbą całkowitą (np. Integer -jak w naszym przypadku). Składnia pętli for wymaga tylko jednej instrukcji po słowie kluczowym do, dlatego (podobnie jak w przypadku if), aby wykonać więcej instrukcji w danej pętli, korzystamy z instrukcji złożonej.

Powyższa instrukcja zlicza w górę, tzn. wartość licznika jest zwiększana za każdym razem o jeden. Jeśli zamiast słowa to użyjemy słowa downto pętla będzie tzw. pętlą malejącą gdyż wartość licznika będzie pomniejszana o jeden. Należy tylko zwrócić uwagę, aby wówczas wartość początkowa była większa od wartości końcowej. Spójrz na przykład:

```
program PO40;
var x,y : Integer;
begin
   for y:=1 to 10 do
   begin
      for x:=10 downto 1 do Write('*');
      Writeln;
   end;
   ReadLn;
end.
```

Wynik:

```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

Powyższy program rysuje kwadrat wypełniony za pomocą gwiazdek. Składa się on z dwóch pętli zagnieżdżonych w sobie. Pierwsza z nich (zewnętrzna) wykona się 10 razy. Za każdym razem wartość zmiennej Y jest zwiększana o jeden. Wewnątrz niej wykonuje się druga pętla (wewnętrzna). Ona również wykona się 10 razy (tyle, że zlicza w dół - od 10 do 1), za każdym razem wyświetlając gwiazdkę. W efekcie powstanie jedna linia złożona z 10 gwiazdek. Kiedy wewnętrzna pętla się zakończy wykona się instrukcja Writeln, która przeniesie kursor do linii następnej. Cała ta operacja się powtórzy (zewnętrzna pętla), a więc narysowana zostanie kolejna linia gwiazdek, później następna, itd.

Spójrz na następny przykład - bardzo podobny do powyższego.

```
program P041;
var x,y : Integer;
begin
   for y:=1 to 10 do
   begin
      for x:=1 to 10 do Write(x*y,'  ') ;
      Writeln;
   end;
   ReadLn;
end.
```

Tym razem wyświetlona zostanie tabliczka mnożenia. Jednak nie jest ona zbyt dobrze czytelna, ponieważ różne elementy owej tablicy mają różną szerokość (z różnej liczby znaków się składają). Co zrobić, aby poszczególne elementy naszej tabliczki były w równych kolumnach? Wewnętrzną pętlę zmień na następującą:

```
for x:=1 to 10 do Write(x*y:4);
```

Program powinien więc wyglądać następująco:

```
program P042;
var x,y : Integer;
begin
   for y:=1 to 10 do
   begin
      for x:=1 to 10 do Write(x*y:4, '  ') ;
      Writeln;
   end;
   ReadLn;
end.
```

Wynik:

```
   i    2    3    4    5    6    7    8    9   10
   2    4    6    8   10   12   14   16   18   20
   3    6    9   12   15   18   21   24   27   30
   4    8   12   16   20   24   28   32   36   40
   5   10   15   20   25   30   35   40   45   50
   6   12   18   24   30   36   42   48   54   60
   7   14   21   28   35   42   49   56   63   70
   8   16   24   32   40   48   56   64   72   80
   9   18   27   36   45   54   63   72   81   90
  10   20   30   40   50   60   70   80   90  100
```

Prawda, że znacznie lepiej?

Kiedy omawialiśmy sposób określania, z jaką dokładnością mają być wyświetlane liczby rzeczywiste, jako pierwszy argument (po dwukropku) pisaliśmy 0 i wspomnieliśmy, że zajmiemy się tym później. Właśnie nastała ta pora.

Otóż ten pierwszy argument określa na ilu miejscach ma być zapisana dana liczba (czy wyraz). Jeśli dana sekwencja do wyświetlenia składa się z mniejszej ilości znaków, zostanie uzupełniona na początku spacjami, tak by zawsze zajmowała określoną liczbę znaków.

W naszym przykładzie napisaliśmy :4, co oznacza, że każdy element tabliczki mnożenia ma się składać z czterech liter. Jeśli więc wynik 1*10=10, to nie wyświetlona zostanie sama ta liczba, ale napis ' 10', a więc na początku umieszczone zostaną dodatkowe spacje. To sprawia, że wszystkie kolumny są równe.

Oto kolejny program wykorzystujący pętlę for:

```
program PO43;
var I,W : Integer;
begin
   W:=1;
   for i:=1 to 10 do
   begin
      W:=W*2;
      Writeln('2^',I,'=',W);
   end;
   ReadLn;
end.
```

```
Wynik:

2^1 = 2
2^2=4
2^3 = 8
2^4=16
2^5=32
2^6=64
2^7=128
2^8=256
2^9=512
2^10=1024
```

Tym razem wyświetlone zostały kolejne potęgi liczby 2.

6.2. Instrukcja REPEAT

Składnia tej instrukcji jest następująca:

```
repeat
  Instrukcja;
  Instrukcja;

until warunek;
```

Wszystkie instrukcje znajdujące się między słowami kluczowymi repeat i until wykonywane są tak długo, aż warunek stanie się prawdziwy. Czytając inaczej - pętla się kończy, gdy warunek zostanie spełniony (przyjmie wartość TRUE).

Spójrz na przykład, w którym pętla została zastosowana:

```
program PO44;
var     R : Real;
     Znak : Char;
begin
   repeat
     Write('Podaj promień koła>');
     ReadLn(R);
     Writeln ('Pole koła wynosi: ',PI*R*R:0:3);
     Writeln('Obwód koła wynosi: ',2*PI*R:0:3);
     Writeln;
     Write('Czy chcesz powtórzyć program? (T/N) > ');
     Read(Znak);
   until UpCase(Znak)o'T';
end.
```

Wynik:

```
Podaj promień koła>2
Pole koła wynosi: 12.566
Obwód koła wynosi: 12.566

Czy chcesz powtórzyć program? (T/N) > T
Podaj promień koła>4
Pole koła wynosi: 50.265
Obwód koła wynosi: 25.133

Czy chcesz powtórzyć program? (T/N) > N
```

W powyższym programie wykorzystaliśmy pętlę repeat...until, aby cały program (który umieściliśmy w jej wnętrzu) wykonywać tyle razy, ile użytkownik sobie życzy.

Kiedy użytkownik proszony jest o udzielenie odpowiedzi na pytanie, czy chce powtórzyć program, znak, który wprowadzi zapamiętywany jest w zmiennej Znak. Użytkownik mógł wcisnąć literkę „t", ale mógł również wcisnąć „T". Ponieważ w przypadku wszelkich warunków w Turbo Pascalu, istotna jest wielkość znaków, zamieniliśmy wprowadzony przez użytkownika znak na wielką literę (korzystając z funkcji UpCase). Dzięki temu, nie jest ważne, czy wprowadził on małe „t", czy wielkie „T", w programie zawsze „widziane" jest ono jako wielkie „T".

Oto kolejny przykład:

```
program P045;
uses Crt;
var G,M,S   Integer;
begin
      =0
      =0
      =0
   repeat
      S:=S+1;
      if S=60 then
      begin
        S:=0;
        M:=M+1;
      end;
      if M=60 then
      begin
        M:=0;
        G:=G+1;
      end;
      GotoXY(5,5
      Writeln(G,     M,   ',S,     ');
      Delay(1000
   until KeyPressed;
end.
```

Powyższy program pokazuje czas działania programu (z dokładnością do sekundy) i wyświetla go w jednym miejscu na ekranie monitora. Wykorzystaliśmy w nim kilka procedur i funkcji, z którymi spotykasz się pierwszy raz.

Druga linijka programu to uses Crt. Jest to tzw. deklaracja modułu, dzięki której możemy wykorzystać kilka nowych procedur. Dokładnie omówiona ona zostanie wtedy, kiedy skupimy się na modułowości programów w Turbo Pascalu. Tymczasem wystarczy byś wiedział, że w tym programie jest ona konieczna.

Jeśli nie zainstalowałeś specjalnego patcha poprawiającego błąd w module Crt i posiadasz szybki procesor (kilkaset MHz) powyższy program może u Ciebie nie działać poprawnie. Dlatego zajrzyj do Dodatku B i dokonaj poprawek w swojej wersji kompilatora.

Program działa w pętli repeat...until. Jednak warunek, podany po słowie unitl wygląda dość specyficznie -jest to funkcja KeyPressed. Zwraca ona wartość typu Boolean, a więc TRUE lub FALSE. Dlatego z niczym jej nie porównujemy. Dopuszczalny byłby zapis: until KeyPressed=TRUE, jednak nie ma potrzeby dodatkowego wzbogacania programu o zbędny kod. Jeśli użytkownik nie wciśnie żadnego klawisza na klawiaturze, funkcja zwraca FALSE. Kiedy natomiast jakiś klawisz zostanie wciśnięty, funkcja zwróci TRUE i wówczas pętla, a zatem i program, zakończy działanie.

Wewnątrz owej pętli zliczamy sekundy w zmiennej S. Kiedy dojdziemy do wartości granicznej (60), zwiększamy minuty - zmienna M, a sekundy ustawiamy na zero. Analogicznie w przypadku minut i godzin.

Procedura Delay(1000), powoduje, że program zatrzyma się na jedną sekundę. To właśnie dzięki niej program zlicza sekundy, gdyż każda kolejna pętla wykonywana jest z tym opóźnieniem. Delay służy to zatrzymywania programu na liczbę milisekund, określoną przez argument. Tysiąc milisekund to jedna sekunda.

Procedura GotoXY służy to określania współrzędnych kursora na ekranie monitora, a więc do określania miejsca, w którym procedury Write iWriteln wyświetlają napisy. W naszym przykładzie, ustawiamy współrzędne kursora na (5,5). Pierwszy argument, to numer kolumny (X) na ekranie, natomiast drugi, to numer wiersza (Y).

Modułami i deklaracją uses zajmiemy się dokładniej w jednym z kolejnych rozdziałów. Jednak, jeśli w którymś z następnych swoich programów, chciałbyś użyć instrukcji Delay lub GotoXY zawsze na górze programu (druga linijka) musisz wpisać Uses Crt.

6.3. Instrukcja WHILE

Składnia tej instrukcji jest następująca:

```
while warunek do instrukcja;
```

Oto przykład:

```
program P046;
var N : String;
begin
   Write('Podaj hasło');
   ReadLn(N);
   while N<>'tajne' do
   begin
       Writeln('Złe hasło!');
       Write('Podaj hasło>');
       ReadLn(N);
   end;
   Writeln('Hasło prawidłowe...');
end.
```

W powyższym programie użytkownik proszony jest o wprowadzenie hasła. Słowo, które wpisze, zapamiętywane jest w zmiennej N. Następnie porównujemy je z określonym przez nas, prawidłowym hasłem. Jeśli jest ono równe 'tajne', pętla while nie wykona się ani razu, a więc wyświetlony zostanie końcowy komunikat. Kiedy jednak warunek zostanie spełniony (wprowadzone hasło będzie inne niż prawidłowe), wykona się instrukcja złożona po słowie do, a więc wyświetlony zostanie stosowny komunikat i użytkownik zostanie ponownie zapytany o hasło.

Jak możemy więc dostrzec - pierwsza różnica pomiędzy pętlą while i repeat jest taka, że ta pierwsza może nie wykonać się ani razu (warunek sprawdzany jest na początku), natomiast pętla repeat wykona się zawsze przynajmniej raz (warunek sprawdzany na końcu).

Druga różnica jest taka, że pętla while wykonuje się tak długo, dopóki warunek jest prawdziwy i kończy się, kiedy przyjmie wartość FALSE, a w przypadku pętli repeat było odwrotnie. Pętla wykonywała się tak długo, dopóki warunek był FALSE i kończyła wtedy, gdy przyjmował wartość TRUE.

6.4. Słowa Break i Continue

Przy korzystaniu z pętli w programach bardzo często zajdzie potrzeba użycia dwóch dodatkowych słów kluczowych, które w znaczny sposób wpływają na ich przebieg.

Słowo kluczowe `Break` sprawia, że wykonywanie danego bloku, wewnątrz instrukcji iteracyjnej, zostanie zakończone i jako następna, wykona się instrukcja znajdująca się za daną pętlą. Często korzystamy z tego słowa, kiedy wewnątrz jakiegoś bloku, wykonującego się wielokrotnie, zachodzi potrzeba natychmiastowego przerwania iteracji.

Słowo kluczowe `Continue` działa podobnie. Po jego napotkaniu również następuje przerwanie wykonywania instrukcji w pętli, jednak nie jest ona kończona tylko następuje skok do instrukcji warunkowej, a więc w przypadku pętli `for` - do warunku `licznik<=wartość końcowa`, dla pętli `while` warunku po tym słowie, a dla pętli `repeat`, do warunku umieszczonego po słowie `until`.

7. Typy danych

O typach danych mówiliśmy już wcześniej. Dlatego wiesz już dobrze, czym one są, w jakim celu zostały wprowadzone i na co wpływają. Potrafisz także tworzyć zmienne poznanych już typów prostych. W tym rozdziale dowiesz się, jak tworzyć własne typy danych, czym są rekordy i tablice, czyli bardziej złożone struktury. Poznamy również instrukcję `with`.

7.1. Typ okrojony

Dotychczas poznaliśmy proste typy danych takie jak `Integer`, `String`, czy `Real`. Użytkownik ma również możliwość tworzenia własnych typów, np.:

```
type
   Liczby = 1..10;
```

W ten sposób stworzyliśmy własny tzw. typ okrojony. Zmienne, które zadeklarujemy jako `Liczby` będą mogły przyjmować wartości z zakresu od 1 do 10 (całkowite).

Definicja własnego typu występuje po słowie kluczowym `type` i musi być umieszczona przed blokiem deklarującym zmienne i po bloku deklarującym stałe (co jednak nie wymogiem).

Spójrz na przykład:

```
program PO47;
type  Typ_Rok=1950..2025;
var Rok : Typ_Rok;
begin
   Write('Podaj rok>');
   ReadLn(Rok);
end.
```

Z typów okrojonych korzystamy często wówczas, gdy piszemy program realizujący konkretne zadanie i mamy np. z góry przewidziane możliwe wartości dla niektórych zmiennych. Nie korzysta się jednak z tego typu danych zbyt często. Z reguły deklarujemy zmienne jako `Integer` lub `Longint` co w zupełności wystarczy.

7.2. Typ zbiorowy

Turbo Pascal umożliwia tworzenie tzw. typów zbiorowych. Są one bardzo podobne do zbiorów rozumianych w ujęciu matematycznym. Zacznijmy od przykładu:

```
type
   Cyfry = set of 0..9;
```

W taki oto sposób utworzyliśmy nowy typ danych o nazwie Cyfry. Zmienne tego typu będą zbiorami, które pozwolą na przechowywanie cyfr. Nie można mylić zbioru i deklaracji typu zbiorowego z typem wyliczeniowym. Zmienne typu wyliczeniowego mogły przyjmować tylko jedną wartość z określonego przedziału. Zbiór może zawierać ich kilka. Spójrz na przykład:

```
program PO48;
type Cyfry = set of 0..9;
var A : Cyfry;
begin
   A:=[1,3,4],•
   if 1 in A then Writeln('l należy do zbioru')
             else Writeln('l nie należy do zbioru');
   ReadLn;
end.
```

Wynik:

```
1 należy do zbioru
```

W powyższym programie utworzyliśmy nowy typ danych, a następnie zadeklarowaliśmy zmienną A tego typu. A to zbiór, który może zawierać cyfry od 0 do 9. W programie, zainicjowaliśmy nasz zbiór, czyli nadaliśmy mu wartość początkową. Zbiór A zawiera więc cyfry 1, 3, 4. W instrukcji warunkowej sprawdzamy, czy cyfra 1 należy do zbioru (znajduje się w nim) i wyświetlamy stosowny komunikat.

Jak widzimy, przy deklaracji typu zbiorowego używamy zwrotu set of, co oznacza „to zbiór". Do zmiennych zbiorowych przypisujemy elementy używając nawiasów kwadratowych. Zapis:

```
A:=[1,2,3];
```

Oznacza, że zbiór A zawiera trzy cyfry - 1, 2 i 3. Żadnych innych elementów nie posiada. Możemy dodać nowy element do zbioru korzystając z operatora sumy „+", oto przykład:

```
A:=A+[9];
```

W ten sposób do naszego zbioru dodaliśmy cyfrę 9. Zawiera więc on już cztery cyfry: 1, 2, 3 i 9. Analogicznie, korzystając z operatora różnicy „-" możemy usunąć wybrany element ze zbioru. Oto przykład:

```
A:=A-[1,5];
```

W powyższej instrukcji chcieliśmy usunąć dwa elementy - cyfry 1 i 5. Usunęliśmy w rzeczywistości tylko jedną- 1, gdyż cyfry 5 w ogóle w nim nie było.

Korzystając z operatora iloczynu „*" - możemy wyznaczyć część wspólną dwóch zbiorów.

Słowo kluczowe in pozwala na sprawdzenie, czy dany element należy do zbioru, czy też nie (spójrz na powyższy przykład). Dokładniej zobrazowane zostało to poniżej:

```
program PO49;
type Cyfry = set of 1..9;
var A,B : Cyfry;
    i   : Integer;
begin
  A:=[2,4,6,8];
  B:=[1,3,5,8];

   A:=A*B;     {zbiór A to część wspólna zbiorów A i B)

   for i:=1 to 9 do
   begin
      if i in A then Writeln('Cyfra ',i,
                             ' należy do zbioru A')
                 else Writeln('Cyfra ',i,' nie <
                              należy do zbioru A');
   end;
   ReadLn;
end.
```

Wynik:

```
Cyfra 1 nie należy do zbioru A
Cyfra 2 nie należy do zbioru A
Cyfra 3 nie należy do zbioru A
Cyfra 4 nie należy do zbioru A
Cyfra 5 nie należy do zbioru A
Cyfra 6 nie należy do zbioru A
Cyfra 7 nie należy do zbioru A
Cyfra 8 należy do zbioru A
Cyfra 9 nie należy do zbioru A
```

Zbiory są bardzo wygodnymi, w niektórych sytuacjach, strukturami **danych.** Znacznie upraszczają zapis pewnych algorytmów. Spójrz na następujący program:

```
program P0 5 0;
type TSpec = set of Char;
var Napis : String;
    Spec  : TSpec;
begin
    Napis:=' *-+  To jest jakiś napis';

    Spec:=['*', '-','+',' '];
    while (Napis<>'') and (Napis[1] in Spec) do <
                              Delete(Napis,1,1);

   Writeln(Napis);
end.
```

Wynik:

```
To jest jakiś napis
```

Przypomnijmy, że zmienne napisowe (String) to uporządkowane ciągi znaków, a za pomocą instrukcji Delete usuwamy fragment tekstu z napisu (w naszym przykładzie pierwszą literę).

Powyższy program działa na tej zasadzie, że usuwa z tekstu wszystkie znaki zdefiniowane w zmiennej Spec. Dokonuje się to w pętli while, która trwa tak długo. dopóki napis nie jest pusty i pierwsza litera należy do wyznaczonego zbioru.

Napiszmy teraz kolejny program. Zadaniem będzie sprawdzenie, ile razy w danym tekście wystąpiły małe litery, ile razy wielkie, ile cyfry, a ile pozostałe znaki. Dzięki zbiorom problem jest niezwykle prosty do rozwiązania.

```
program P0 5 1;
var Napis       : String;
    M,W,C,I,K   : Integer;
begin
   Write('Podaj napis: ');
   ReadLn(Napis);
   M:=0;      {małe litery}
   W:=0;      {wielkie litery}
   C:=0;      {cyfry}
   I:=0;      {inne}
```

```
    for k:=1 to Length(Napis) do
      if Napis[k] in ['a'..'z'] then Inc(M)
      else if Napis[k] in ['A'..'Z'] then Inc(W)
      else if Napis[k] in ['0'..'9'] then Inc(C)
      else Inc(I) ;

    Writeln('Podsumowanie') ;
    Writeln('Małe litery ':20,M);
    Writeln('Wielkie litery ':20,W);
    Writeln('Cyfry: ':20,C);
    Writeln('Inne znaki: ':20,I);
    Readln;
end.
```

Przykładowy wynik:

```
Podaj napis: Mam 21 lat!!
Podsumowanie
       Małe litery 5
    Wielkie litery 1
            Cyfry: 2
       Inne znaki: 4
```

Powyższy program wymaga pewnego wyjaśnienia. Wykorzystaliśmy w nim nową instrukcję Inc, aby program był bardziej przejrzysty. Powoduje ona zwiększenie zmiennej całkowitej przekazanej jej jako argument o jeden. Przykładowo - dla X=1, wywołanie Inc(X) jest równoważne zapisowi X:=X+1, czyli na koniec zmienna przyjmie wartość 2. Analogiczną do tej funkcji (lecz nie użytą w programie) jest instrukcja Dec, która zmniejsza wartość zmiennej o jeden.

W powyższym programie stworzyliśmy cztery zmienne określone jako całkowite. Są to odpowiednio: M - licznik wystąpień małych liter; W - licznik wystąpień wielkich liter; C - licznik wystąpień cyfr; I - licznik wystąpień pozostałych znaków. Na początku, po wprowadzeniu tekstu przez użytkownika i zapamiętaniu go pod zmienną Napis zerujemy te zmienne. Po tym następuje wykonanie pętli for. Przebiega ona od wartości 1 do Length(Napis), a więc do takiej wartości, z ilu liter składa się napis. W pętli zwiększana jest zmienna licznikowa k. Instrukcja wykonywana w pętli, to warunek if. Sprawdzane jest w nim, czy k-ta litera napisu należy do określonego zbioru (małych liter). Jeśli tak, powiększana jest wartość zmiennej M. Jeśli nie należy do tego zbioru, jako instrukcja po else wykonuje się ponownie instrukcja warunkowa. Tym razem sprawdzamy, czy ta sama, k-ta litera należy do zbioru wielkich liter. Jeśli tak, zwiększamy wartość zmiennej W, jeśli nie - analogicznie sprawdzamy czy należy do zbioru cyfr. Jeśli i do nich nie należy, zwiększamy zmienną I - czyli inne znaki.

Zbiorów nie określaliśmy w odrębnych zmiennych, bo nie miało to większego sensu. Określaliśmy je bezpośrednio w treści programu. Nie definiowaliśmy ich poprzez wymienienie wszystkich możliwych znaków (co byłoby oczywiście poprawne), lecz korzystając z wyliczenia, tzn. wielokropka.

7.3. Rekordy

Pascal pozwala również na tworzenie tzw. rekordów. Rekord to również nowy typ danych. Każda zmienna rekordowa składa się z pewnej ilości pól. Każde pole z kolei ma swój identyfikator i - oczywiście - własny typ. Oto przykład deklaracji typu Dane:

```
type
   Dane = record
    Imie     : String;
    Nazwisko : String;
    Telefon  : Integer;
end;
```

W ten sposób stworzyliśmy własny typ danych o nazwie Dane. Zmienne zadeklarowane jako tego typu będą posiadały „w sobie" trzy pola. Dwa napisowe i jedno, jako liczbę całkowitą. Dostęp do tych pól uzyskujemy dzięki operatorowi kropki. Spójrz na przykład:

```
var Osoba : Dane;

begin
   Osoba.Imie:='Jan';
   Osoba.Nazwisko:='Nowak';
   Osoba.Telefon:=62423 23;

end.
```

Rekordy są strukturami danych bardzo często wykorzystywanymi. Szczególnie w połączeniu z tablicami (o których powiemy później) tworzą potężne struktury wyśmienicie nadające się do tworzenia baz danych. Jednak również pojedyncze zmienne -jako rekordy - często sprawiają, że program staje się bardziej czytelny.

Napiszmy teraz kolejny program. Będzie to odbijający się od krawędzi ekranu napis. Oto jak mógłby on wyglądać:

```
program P052;
uses Crt;
const Napis = 'Turbo Pascal'
type TPunkt = record
       x   Integer;
       Y   Integer;
       a   Integer;
       b   Integer;
     end;
var P : TPunkt;
    i : Integer;
begin
   P.X:=5;
   P.Y:=5;
   P.A:=1;
   P.B:=1;
   repeat
      GotoXY(P.X,P.Y);
      Write(Napis);
      Delay(lOO);
      GotoXY(P.X,P.Y);
      for i:=l to Length(Napis) do Write(' ') ;
      Inc(P.X,P.A);
      Inc(P.Y,P.B);
      if (P.X<=1) or (P.X>=80-Length(Napis) ) then <
                                          P.A:=-P.A;
      if (P.Y<=1) or (P.Y>=25) then P.B:=-P.B;
   until KeyPressed;
end.
```

Przeanalizujmy wspólnie powyższy program, choć jak zapewne zauważyłeś, nie pojawiło w nim się nic zupełnie nowego.

Zadeklarowaliśmy w nim moduł Crt (zapis uses Crt), aby móc skorzystać z poznanych już wcześniej instrukcji GotoXY (służącej do ustawiania kursora na ekranie) i Delay (zatrzymującej program na pewien czas). Napis, który chcemy aby się odbijał od krawędzi ekranu umieściliśmy w stałej Napis. Dzięki temu możemy w każdej chwili zmienić tę treść na inną, a program będzie poprawnie funkcjonował.

Stworzyliśmy własny typ danych o nazwie TPunkt i zmienną P zadeklarowaną tego typu. W owej zmiennej będziemy przechowywać współrzędne wyświetlanego napisu (X i Y) i tzw. skoki (A i B). Bowiem, aby przesunąć nasz napis zwiększymy po prostu odpowiednio współrzędną X o skok A i współrzędną Y o skok B. Nasze zmienne A i B będą przyjmowały wartości 1 i -1. Kiedy, np. A będzie równe 1, to zwiększając współrzędną X o A będzie się ona zwiększać właśnie o 1. Kiedy skok A przyjmie wartość -1, zwiększenie (poprzez Inc) zmiennej X o wartość A spowoduje tak naprawdę zmniejszenie jej wartości.

Tak też się dzieje w naszym programie. Najpierw wprowadzamy ustawienia początkowe. Następnie wykonywana jest pętla repeat...until KeyPressed, czyli trwająca tak długo, aż użytkownik wciśnie jakiś klawisz na klawiaturze. Wewnątrz owej pętli ustawiamy współrzędne kursora w miejscu (X,Y) i wyświetlamy nasz napis. Następnie zatrzymujemy na chwilę obraz, aby użytkownik mógł go dostrzec (poprzez instrukcję Delay) i zmazujemy napis (aby nie zostawał ślad). Zmazanie napisu odbywa się w ten sposób, że najpierw ponownie ustawiamy kursor we wskazanym miejscu, a następnie - poprzez pętlę for - wyświetlamy tyle spacji, z ilu liter napis się składa. Kiedy już tekst zostanie z ekranu zmazany zwiększamy współrzędne X i Y o skoki A i B (lub zmniejszamy, jeśli A lub B mają wartości ujemne). Aby napis odbijał się od krawędzi ekranu zmienne skokowe musimy odpowiednio korygować. Kiedy zmienna A miała wartość 1 (na początku) napis poruszał się w prawą stronę. Poprzez warunek if sprawdzamy, czy napis nie doszedł do prawej krawędzi (80-długość napisu) i jeśli tak, zmieniamy wartość skoku na przeciwną, a więc na -1. Od tego momentu napis zacznie poruszać się w lewą stronę. W tym samym warunku sprawdzamy, czy napis osiągnął lewy koniec, a jeśli tak - również zmieniamy skok na przeciwny, czyli na 1. Analogicznie czynimy w przypadku współrzędnej Y.

7.3.1. Instrukcja WITH

Czasami się zdarza, że zmienna typu rekordowego ma dość długi identyfikator, a w naszym programie wielokrotnie z niej korzystamy (np. w operacjach arytmetycznych). Wówczas, aby nie pisać cały czas nazwy pola oddzielonego kropką, co w znaczniej mierze sprawia, że kod programu staje się mniej zrozumiały, od nazwy zmiennej - możemy skorzystać z tzw. instrukcji wyłuskania - with. Oto przykład:

```
with Osoba do
begin
  Imie:='Karol';
  Telefon:=6242357;
end;
```

albo

```
with P do
    X := 5;
    Y := 5;
    A := 1;
    B := 1;
end;
```

Dzięki powyższemu zapisowi uzyskaliśmy dostęp do poszczególnych pól zmiennych Osoba i P bez podawania ich przedrostka. Spójrz na poniższy fragment programu, który pokazuje jak czytelność kodu źródłowego można zachować właśnie dzięki tej konstrukcji.

```
type
    Dane = rekord
        Stypendium_naukowe : Real;

   end;

var Osoba : Dane;

Osoba.Stypendium_naukowe:=Osoba.Stypendium_Naukowe+-*
                  (Osoba.Stypendium_naukowe*10)/100;
```

Jeśli skorzystalibyśmy z operatora with ostatnia linijka wyglądałaby tak:

```
with Osoba do
  Stypendium_naukowe:=Stypendium_Naukowe+-*
                      (Stypendium_naukowe*10)/100;
```

i zapis stanie się bardziej czytelny.

Nazwa pola Stypendium_naukowe jest oczywiście całkowicie poprawna, z punktu widzenia kompilatora Turbo Pascala. Staraj się jednak unikać tak długich identyfikatorów gdyż - jak mogłeś zobaczyć powyżej - nawet korzystając z instrukcji with program przestaje być w pełni czytelny. Trzeba jednak pamiętać, aby nazwa zmiennej kojarzyła się z treścią, jaka ma być w niej przechowywana. Kieruj się również zasadą, że spółgłoski są cenniejsze niż samogłoski (w nazewnictwie zmiennych). Stąd też dobrym rozwiązaniem byłoby nazwanie powyższego pola StypNauk. Również wiadomo, co kryje się pod zmienną i jednocześnie zapis jest bardziej zwięzły.

7.4. Tablice

Tablica to również nowy typ danych, który możemy stworzyć korzystając ze słowa kluczowego array. Jest to ciąg zmiennych tego samego typu ułożonych kolejno jeden za drugim. Dostęp do poszczególnych pól tablicy uzyskujemy przez indeks przekazany w nawiasach kwadratowych. Spójrz na przykład:

```
type Tablica : array[1..10] of Integer;

var T : Tablica;

T[1]:=23;
T[4]:=87;
```

Powyżej utworzyliśmy nowy typ danych o nazwie Tablica, a dalej: zmienną T nowego typu. Nasza własna struktura danych jest tablicą, o czym świadczy słowo kluczowe array po dwukropku. Składa się ona z 10 elementów ułożonych jeden za drugim i ponumerowanych od 1 do 10. Każdy taki element jest liczbą całkowitą, co określa nam typ Integer. Dostęp do poszczególnych składowych tablicy uzyskujemy przez indeks (numer elementu) umieszczony w nawiasach kwadratowych.

Czy czegoś Ci to nie przypomina? Kiedy omawialiśmy typ String wspomniałem, że jest to uporządkowany ciąg zmiennych typu Char. Jest to więc tablica znaków (Char). Dostęp do poszczególnych elementów (liter) uzyskiwaliśmy poprzez nawias kwadratowy. Ma to miejsce również i w tym przypadku. Jednak typ String jest tablicą specyficzną, na której można wykonywać operację łączenia napisów (+), sprawdzić długość napisu (Length), wyciąć lub usunąć fragment z tekstu (Copy i Delete). W przypadku tablic zadeklarowanych słowem array nie mamy żadnych funkcji operujących na strukturze, jako na całości. Nie możemy więc korzystać z wiadomości, poznanych przy omawianiu typu String.

Tablica może mieć elementy dowolnego typu, np. napisy (String), liczby rzeczywiste (Real), czy też wartości logiczne (Boolean). Spójrz na kolejny fragment programu:

```
type Tablica : array[3..13] of Boolean;
var T : Tablica;
    I : Integer;
begin
   for i:=3 to 13 do T[i]:=FALSE;

end'."
```

Utworzyliśmy nowy typ danych: tablicę 11 elementów numerowanych od 3 do 13. Każdy element jest typu Boolean. Następnie - w treści operacyjnej programu - nadaliśmy każdemu z pól wartość logiczną FALSE. Zauważ, że ta struktura numerowana jest od liczby 3. Pierwszy element ma numer 3 i nie jest możliwe odwołanie T [1] : = . . ., bowiem nie ma w tablicy elementu o indeksie (numerze) 1.

Jeśli chcemy utworzyć zmienną typu tablicowego nie musimy koniecznie tworzyć najpierw nowego typu, a następnie deklarować taką zmienną. Obie te czynności możemy wykonać w jednym miejscu, np.:

```
var T : array[1..10] of String;
```

Podobnie z każdym innym typem. Na przykład:

```
var Osoba : record imie: string;wiek:Integer,-end;
```

Napiszmy program, który poprosi użytkownika o wprowadzenie pewnej ilości imion, które zapamięta w tablicy. Następnie użytkownik zostanie zapytany o kolejne imię i program stwierdzi, czy znajduje się ono w jego bazie, czy też nie.

```
program PO53;
const Max = 10;
var Imiona : array[1..Max] of String;
    Szukaj : String;
    i      : Integer;
    Znaleziono : Boolean;
begin
   for i:=1 to Max do
   begin
      Write('Podaj ',i,' imię >');
      ReadLn(Imiona[i]);
   end;
   Writeln('Wyszukiwanie...');
   repeat
      Write('Podaj szukane imię (ENTER - koniec) >');
      ReadLn(Szukaj);
      Znaleziono:=FALSE;
      for i:=1 to Max do
         if Imiona[i]=Szukaj then Znaleziono:=TRUE;
      if Znaleziono then
            Writeln(' Znaleziono to imię.')
       else Writeln(' Brak tego imienia.');
   until Szukaj='';
end.
```

W powyższym programie zadeklarowaliśmy tablicę o nazwie Imiona. Składa się ona z Max pól, a więc z takiej liczby, jaką wartość przypiszemy do tej stałej. Na początku użytkownik proszony jest o wprowadzenie kolejnych imion, które zapamiętywane są w przygotowanej strukturze. Następnie użytkownik proszony jest o wprowadzenie szukanego imienia. Tekst, który wpisze zapamiętujemy w zmiennej szukaj. Poprzez pętlę for porównujemy zmienną szukaj z wszystkimi elementami tablicy. Jeśli znajdziemy nasze imię, zmiennej Znaleziono nadajemy wartość TRUE. Jeśli nie odnajdziemy imienia, na koniec zmienna będzie posiadała wartość FALSE. Korzystając z instrukcji warunkowej if, sprawdzamy następnie, jaką wartość ma ta zmienna i wyświetlamy stosowny komunikat. Blok szukający umieszczony jest w pętli repeat i trwa tak długo, aż użytkownik nie wpisze imienia, tylko wciśnie sam klawisz [ENTER].

A teraz jeszcze jeden program. Tym razem jego zadaniem będzie wylosowanie liczb całkowitych, a następnie odszukanie elementu największego w tablicy.

```
program PO54;
const Ilosc = 20;
var T : array[1..Ilosc] of Integer;
    i,max : Integer;
begin
   Randomize;
   Writeln('Losowanie liczb...');
   for i:=1 to Ilosc do T[i]:=Random(100);
   Writeln('Wyświetlanie liczb...');
   for i:=1 to Ilosc do
         Writeln('Liczba nr. ',i,' to ',T[i]);
   Writeln('Szukanie maksimum...');
   Max:=T[1];
   for i:=2 to Ilosc do
       if T[i]>Max then Max:=T[i];
   Writeln('Maksimum to: ',Max);
   ReadLn;
end.
```

W programie przytoczonym wyżej skorzystaliśmy z nieznanej dotychczas instrukcji Random. Jest to funkcja, która zwraca liczbę losową (całkowitą). Przedział, z którego losujemy liczby określamy argumentem. Precyzując - argument, który przekażemy tej funkcji określa spośród ilu liczb (począwszy od zera) ma być dokonane losowanie. Wywołanie Random(100) sprawia, że

wylosowana zostanie liczba z zakresu od 0 do 99 (czyli spośród 100 liczb). Random(50), to losowanie z przedziału od 0 do 49. Gdybyśmy chcieli wylosować liczbę z przedziału od 1 do 10, napisalibyśmy Random(10)+1.

Losowanie to tak naprawdę wykonanie pewnych operacji matematycznych. We wzorze, z którego wyliczana jest liczba kolejna (funkcją Random) potrzebny jest pewien argument-zmienna. Jeśli miałby on wartość stałą, wyliczanie liczb losowych przebiegałoby zawsze w ten sam sposób, tzn. pojawiałyby się „losowe" liczby, ale zawsze takie same. Dlatego w każdym programie, który wykorzystuje instrukcję Random musi się też znaleźć instrukcja Randomize (tylko jeden raz, np. na początku). Sprawia ona, że ten „zalążek", czyli argument wykorzystywany przy losowaniu będzie za każdym razem inny. Aby to dokładnie zobaczyć i zrozumieć usuń linijkę Randomize z przytoczonego przykładu i uruchom kilka razy program. Zobaczysz, że liczby za każdym razem będą takie same.

Odszukiwanie elementu największego w tablicy przebiega w następujący sposób. Na początku zakładamy, że wartość największą ma pierwszy element tablicy. Następnie - korzystając z pętli for - przeglądamy wszystkie następne elementy (a więc od 2 do Ilosc) i każdy z nich porównujemy ze zmienną Max. Jeśli któryś z nich będzie od niej większy, zmiennej Max nadajemy jego wartość.

7.4.1. Sortowanie tablic

Sortowanie tablicy jest to pewne jej uporządkowanie w zdefiniowany przez nas sposób. Przykładowo - jeśli mamy w tablicy liczby, możemy je poukładać od najmniejszej do największej (lub odwrotnie). Jeśli mamy napisy (np. nazwiska osób) możemy je uporządkować według alfabetu.

Jest wiele różnych algorytmów (sposobów) sortowania tablic. My nauczymy się tzw. sortowania bąbelkowego, gdyż jest to metoda bardzo prosta w nauce i całkowicie wystarczająca na tym etapie programowania. Zanim jednak do tego przejdziemy należy omówić dwie ważne kwestie.

Jak zamienić między sobą dwie wartości w dwóch zmiennych, tzn. żeby zmienna A zawierała to, co w danej chwili zawiera zmienna B i odwrotnie? Spójrz:

```
A:=B;
B:=A;
```

Czy takie rozwiązanie jest właściwe? Oczywiście nie. Załóżmy, że zmienna A zawiera liczbę 1, a zmienna B 2. Kiedy w pierwszej linijce przypiszemy zmiennej A wartość zmiennej B, zawierać ona będzie liczbę 2. Dlatego później przypisując zmiennej B wartość A (czyli 2) nie dokonamy zamiany. W pierwszym kroku tracimy wartość spod zmiennej A. Dobrym rozwiązaniem jest poniższy przykład:

```
A:=1;
B:=2;
{zamiana}
T:=A;      {T=1}
A:=B;      {A=2}
B:=T;      {B=1}
```

Wykorzystaliśmy dodatkową zmienną T, w której zapamiętaliśmy wartość, jaką traciliśmy w poprzednim przykładzie.

Teraz drugi problem. Wiesz zapewne jak sprawdzić, która z dwóch liczb jest większa, a która mniejsza. Wystarczy posłużyć się operatorem mniejszości lub większości. A jak sprawdzić, który napis jest dalej według alfabetu, a który bliżej?

Rozwiązanie jest również proste i analogiczne. Wystarczy posłużyć się tymi samymi operatorami. Kompilator, kiedy napotka porównanie dwóch napisów z użyciem operatora mniejszości lub większości, zawsze sprawdzi, który z nich jest niżej (wyżej) według kolejności alfabetycznej.

Wyjaśniwszy te dwie kwestie możemy napisać algorytm sortowania bąbelkowego. Algorytm taki jest bardzo prosty i polega na odszukaniu elementu najmniejszego w tablicy i umieszczeniu go na samym jej początku. Następnie - ponowne odszukanie elementu najmniejszego (nie uwzględniając pierwszego) i również umieszczenie go na początku. Spójrz na szablon algorytmu:

```
for i:=1 to ilość-1 do
   for j:=i+1 to ilość do
     if T[i]>T[j] then Zamień i z j.
```

Przeanalizujmy. T - to nasza tablica z pewnymi danymi. Ilość - to liczba elementów w tej tablicy.

Na początku mamy pętlę ze zmienną i od 1 do ilość - 1. Czyli na starcie i = 1. Dla tego przypadku wykonuje się kolejna pętla od i + 1, czyli od 2 do ilość. Także zmienną j będzie „przebiegać" po wszystkich indeksach począwszy od drugiego. Za każdym razem porównujemy element j-ty z i-tym, a więc drugi z pierwszym, trzeci z pierwszym, czwarty z pierwszym, itd. aż wreszcie ostatni z pierwszym. Jeśli się tak zdarzy (warunek if), że pierwszy element będzie większy od któregoś z następnych (albo inaczej - znajdzie się element mniejszy od pierwszego) - zamieniamy je miejscami, czyli jako element pierwszy umieścimy ten mniejszy, a na jego miejsce ten, który dotychczas był pierwszy. W efekcie po - jednym „przebiegu" wewnętrznej pętli na samym początku tablicy (i=1) znajdzie się element najmniejszy. Wykona się wówczas drugi krok pętli zewnętrznej, a więc i przyjmie wartość 2. Pętla z j będzie teraz przebiegać od j = i + 1, czyli od 3 do ilość. Analogicznie - element najmniejszy umieszczony zostanie na drugiej

pozycji. Później i zwiększymy do 3 i w tym miejscu znajdzie się element najmniejszy z pozostałych, itd. Na końcu tablica będzie uporządkowana rosnąco. Jeśli zmienimy znak „>" w warunku if na „<" sortowanie będzie malejące.

Oto konkretny przykład:

```
program P055;
const Ilosc =10;
var T : array[1..Ilosc] of Integer;
    W : array[1..Ilosc] of Integer;
    i,k,P : Integer;
begin
  Randomize;
  for i:=1 to Ilosc do
  begin
     T[i]:=Random(100);
     W[i]:=T[i];
  end;
  for i:=1 to Ilosc-1 do
    for k:=i+1 to Ilosc do
      if W[i]>W[k] then
      begin
          P:=W[i];
          W[i]:=W[k];
          W[k]:=P;
      end;
  for i:=1 to Ilosc do
    Writeln(i:3,'  ',T[i]:5,'  ',W[i]:5);
  .Readln;
end.
```

Przykładowy wynik programu:

```
  1     76     13
  2     36     18
  3     18     18
  4     85     36
  5     53     53
  6     55     55
  7     18     76
  8     90     85
  9     13     86
 10     86     90
```

W powyższym programie zadeklarowaliśmy dwie identyczne tablice T i W. Następnie wylosowaliśmy - do pierwszej z nich - losowe wartości i skopiowaliśmy je do drugiej. W efekcie zawierały one dokładnie te same liczby i w tej samej kolejności. Korzystając z opisanego uprzednio algorytmu sortowania bąbelkowego poukładaliśmy rosnąco liczby z tablicy W, a następnie wyświetliliśmy wyniki w trzech kolumnach.

Postawmy sobie teraz następne zadanie. Chcemy stworzyć małą bazę danych, tak, aby przechowywała nazwiska i wiek 10 naszych znajomych. Na koniec chcemy wszystkie dane wyświetlić w kolejności alfabetycznej. Jak tego dokonać? Jak przechować nazwiska i wiek 10 osób (to problem pierwszy). Możemy stworzyć dwie tablice - jedną z napisami i drugą z wiekiem. Jednak znamy metodę sortowania bąbelkowego, która pozwala na sortowanie tylko jednej tablicy danych. Gdybyśmy posortowali oddzielnie lata i oddzielnie nazwiska, dane nie zgadzałyby się ze sobą. Osoba o nazwisku zaczynającym się na A (pierwsza w tablicy nazwisk) mogłaby mieć najwięcej lat (ostatnia w tablicy wieków). Takie rozwiązanie odpada. Jak inaczej? Wykorzystać rekordy! Tablice mogą składać się z pól dowolnego innego typu, a więc mogą zawierać rekordy. Spójrz na przykład:

```
program P05 6;
const  Ilosc=10;
type TDane = record
       Nazwisko : String[30];
       Wiek     : Byte;
     end;
var Dane  : array[1..10] of TDane;
    i,k   : Integer;
    P     : TDane;

begin
   for i:=1 to Ilosc do
   begin
      Writeln('Osoba numer ',i,'.');
      Write('Nazwisko:     ');
      ReadLn(Dane[i].Nazwisko);
      Write('Wiek: ');
      ReadLn(Dane[i].Wiek);
   end;

   Writein('Sortuję dane...');
   for i:=1 to Ilosc-1 do
     for k:=i+1 to Ilosc do
       if Dane[i] .NazwiskoDane[k] .Nazwisko then
```

```
        begin
          P:=Dane[i];
          Dane[i]:=Dane[k];
          Dane[k]:=P;
        end;

    Writeln('Wyświetlanie danych...');
    for i.:=1 to Ilosc do
       Writeln(i:3,'  ',Dane[i].Nazwisko:33,
                   Wiek:  ',Dane[i].Wiek);

    ReadLn;
end.
```

7.4.2. Tablice wielowymiarowe

Wspomniałem, że przy deklaracji tablic po słowie of możemy umieścić dowolny typ danych i wówczas tablica będzie zawierała elementy tego typu. Robiliśmy tak z liczbami, napisami, a nawet rekordami. Co się jednak stanie, gdy jako typ składowy tablicy określimy... tablicę? Uzyskamy tzw. tablicę wielowymiarową. Spójrz:

```
type
   Tab1 = array[1  .10] of Integer;
   Tab2 = array[1  .10] of Tab1;
var T : Tab2;
begin
    T[1][2]:=2;

end.
```

Stworzyliśmy nowy typ danych o nazwie Tab1. Jest to tablica 10 liczb całkowitych. Do elementów takiej tablicy odwoływalibyśmy się przez nawiasy kwadratowe, np. P[2] :=12. W naszym powyższym programie utworzyliśmy jednak jeszcze drugi typ danych - Tab2. To również tablica 10-elementowa, jednak każdy z jej elementów to., tablica typu Tab1. W ten sposób odwołując się, np. T [1] odwołujemy się do elementu pierwszego, czyli tablicy pierwszej, T [2] do elementu drugiego - tablicy drugiej. Ponieważ do poszczególnych elementów odwołujemy się przez indeks w nawiasach kwadratowych, aby wskazać konkretnie element musimy określić numer tablicy i numer elementu w danej tablicy, np. T[2][3]:=1;.

Tablicę jednowymiarową (poznaną wcześniej) można sobie wyobrazić jako swego rodzaju tabelkę, która ma jeden wiersz i pewną liczbę ponumerowanych kolumn. Tablica dwuwymiarowa (poznana przed chwilą) to tabelka, która ma pewną liczbę wierszy (tablic jednowymiarowych) i pewną liczbę kolumn (tablice jednowymiarowe mają swoje kolumny). Stąd, odwołując się do pewnego pola takiej tablicy dwuwymiarowej, musimy określić dwie współrzędne, tzn. numer wiersza i numer kolumny.

Spójrz na przykład:

```
program P057;
type TWiersz = array[1..10] of Integer;
     TTab    = array[1..10] of TWiersz;
var Tab : TTab;
    i,j : Integer;
begin
   for i:=1 to 10 do
     for j:=1 to 10 do Tab[i] [j] :=i*j ;

   for i:=1 to 10 do
   begin
     for j:=1 to 10 do Write(Tab[i][j] :3) ;
     Writeln;
   end;

   ReadLn;
end.
```

W analogiczny sposób można tworzyć struktury bardziej zaawansowane, np. tablice trójwymiarowe. Z takimi tworami spotykamy się jednak znacznie rzadziej.

Turbo Pascal pozwala na jeszcze inny sposób tworzenia tablic i inną metodę odwoływania się do poszczególnych elementów. Oto przykład:

```
var T : array[1..10,1..10] of Integer;
```

W ten sposób stworzyliśmy również tablicę dwuwymiarową. Jest to krótszy sposób powyższego zapisu, jednak nie do końca równoważny.

Do poszczególnych elementów możemy odwoływać się jeszcze w taki sposób:

```
T[2,3]:=4;
```

Podajemy, więc kolejne wymiary tablicy w jednych nawiasach kwadratowych, oddzielając je tylko przecinkiem. Jest to metoda częściej stosowana z uwagi na prostszy i krótszy (chociaż tylko o jeden znak) zapis.

8. Procedury i funkcje

Kiedy omawiałem i wprowadzałem nowe instrukcje, np. Write, Writeln, Random, itd. często używałem sformułowań „procedura Write", czy też „funkcja Random". Wówczas czytałeś to zwracając szczególną uwagę na identyfikator, a nie na określenie jakiego użyłem, gdyż mogłeś nie wiedzieć, o czym tak naprawdę mówimy. W tym rozdziale zostanie to uściślone i wyjaśnione. Dowiesz się, czym dokładnie jest procedura, a czym funkcja. Nauczysz się tworzyć samemu takie elementy i korzystać z nich.

8.1. Procedury

Procedura to pewien podprogram opatrzony własnym identyfikatorem (nazwą). Może posiadać własny blok deklarujący stałe i zmienne. Zawiera również ciąg instrukcji, które ma wykonać. Jest to, więc zamknięty i zwarty blok instrukcji wykonujący określone działanie. Procedura może oczekiwać pewnych argumentów, które umieszczamy w nawiasie, podczas wywołania i które mogą wpływać na przebieg działania. Write, Writeln, Read, ReadLn, Randomize, GotoXY - to wszystko są procedury. Nazwy, które wymieniłem są ich identyfikatorami i wpisując je w treści programu (z ewentualnymi argumentami) powodujemy, że zostają wykonane. Każda z tych procedur ma określone działanie. Jedne wyświetlają napisy, inne pobierają dane z klawiatury, jeszcze inne ustawiają kursor na ekranie monitora. Programista ma możliwość tworzenia własnych procedur i korzystania z nich w swoich programach. Oto jak wygląda prosty szablon procedury:

```
procedure nazwa;
begin

end;
```

Wewnątrz procedury, w bloku instrukcyjnym (pomiędzy słowami begin i end) umieszczamy ciąg instrukcji, które mają zostać wykonane.

Spójrz na przykład:

```
program PO 5 8 ;
procedure  Napiszlmie;
begin
    Writeln('Kamil');
end;
{ program główny }
begin
   Napiszlmie;
end.
```

Jak widzisz powyżej, procedury umieszczamy przed programem głównym, rozpoczynającym się słowem begin. Jednak po uruchomieniu programu, wykonanie rozpoczyna się oczywiście od ostatniego begin - czyli od programu głównego. W naszym przykładzie stworzyliśmy własną procedurę, którą nazwaliśmy Napiszlmie. Posiada ona jedną instrukcję, która poprzez instrukcję Writeln wyświetla imię „Kamil". W programie głównym umieściliśmy identyfikator naszej procedury, co sprawia, że nastąpi w tym miejscu jej wykonania.

Procedury mogą potrzebować dodatkowych argumentów przy wywołaniu (jak np. Read, ReadLn, czy też Write). Listę deklarującą wymagane argumenty umieszczamy w nagłówku procedury. Spójrz na szablon:

```
procedure nazwa(zmienna : typ; zmienna2 : typ2; . . . ) ;
begin


end;
```

Jak widać listę tę umieszczamy po nazwie procedury w nawiasach okrągłych. Kolejne pozycje oddzielamy średnikiem. Każdy argument ma swoją nazwę i typ danych oddzielone dwukropkiem.

Spójrz na nasz kolejny przykład:

```
program P05 9;
uses Crt;
procedure Wyświetl(X,Y : Integer;Napis : String);
begin
   GotoXY(X,Y);
   Write(Napis);
end;
begin
   Wyświetl(5,6,'Turbo  Pascal');
   ReadLn;
end.
```

W powyższym programie stworzyliśmy własną procedurę o nazwie Wyświetl. Aby ją wywołać, trzeba podać trzy argumenty. Pierwsze dwa mają być liczbami całkowitymi (Integer), a ostatni napisem. W programie głównym wywołujemy naszą procedurę i przekazujemy jej argumenty 5, 6 i napis „Turbo Pascal".

Definiując argumenty w nagłówku procedury tworzymy jakby zmienne lokalne. Kiedy wywołaliśmy naszą procedurę z powyższymi argumentami, zmienna X przyjęła wartość 5, zmienna Y - wartość 6, natomiast zmienna Napis zawierała przekazaną treść. W ten sposób, wewnątrz procedury skorzystaliśmy z tych wartości ustawiając kursor w odpowiednim miejscu i wyświetlając stosowny komunikat.

8.2. Funkcje

Funkcje są bardzo podobne do procedur. To również podprogramy wykonujące określony ciąg instrukcji, opatrzone identyfikatorem, listą argumentów w nagłówku, blokiem begin i end;. Jednak funkcje pozwalają na zwracanie pewnych wartości. Możemy więc je przyrównać do pewnych zmiennych, czy spróbować wyświetlić. Np. funkcja Random skutkowała wylosowaniem liczby z przedziału określonego argumentem. Zwracała wynik, czyli wylosowaną wartość. Dlatego mogliśmy napisać x:=Random(5); czy też Writein(Random(5));. Oto budowa typowej funkcji:

```
function nazwa(zmiennal : typ; zmienna2 : typ; ...) <
                    : typ;
begin

   nazwa:=wartosc;
end;
```

Funkcję rozpoczynamy słowem kluczowym function, po którym umieszczamy jej identyfikator. Dalej - podobnie jak w przypadku procedur - umieszczamy w nawiasie listę argumentów. Jeśli funkcja nie oczekuje argumentów pomijamy ten blok. Po nim, po znaku równości określamy typ zwracanej przez funkcję wartości. Kompilator musi bowiem wiedzieć, czy dana funkcja zwraca liczby, napisy, czy może rekordy. Pomiędzy słowami begin i end; umieszczamy ciąg instrukcji do wykonania. Wartość, która ma zostać zwrócona przez funkcję, przypisujemy do jej identyfikatora. Dla przykładu, funkcja podnoszącą dowolną liczbę do kwadratu mogłaby wyglądać tak:

```
function kwadrat(x : Real)  : Real;
begin
   kwadrat:=x*x;
end;
```

Oczekuje ona jednego argumentu, który jest liczbą rzeczywistą i - w wyniku swojego działania - zwraca również liczbę rzeczywistą. Dokładniej rzecz ujmując, wynik wyrażenia x*x. W swoim programie mógłbyś skorzystać z tej funkcji np. w taki sposób:

```
Writeln(kwadrat(5));
```

Stwórzmy teraz funkcję Silnia. Mogłaby ona przyjąć następującą postać:

```
function silnia(x : Byte) : Word;
var w : Integer;
begin
   w:=1;
   for i:=1 to x do w:=w*i;
   silnia:=w;
end;
```

Powyższa funkcja ma identyfikator silnia. Oczekuje jednego argumentu - małej liczby całkowitej nieujemnej (Byte). Jako wynik również zwraca liczbę całkowitą nieujemną, ale o większym zakresie (Word). Przekazany argument zapamiętywany jest w lokalnej zmiennej x. W funkcji zadeklarowaliśmy jeszcze jedną zmienną (tak jak w programie głównym - słowo var przed słowem begin) o nazwie w.

8.3. Widoczność zmiennych

Zmienne umieszczone wewnątrz nagłówka procedury lub funkcji oraz zmienne zadeklarowane wewnątrz danego podprogramu (poniżej nagłówka i powyżej słowa begin), to tzw. zmienne lokalne. Są one „widziane" przez kompilator tylko w obrębie danego podprogramu w przeciwieństwie do zmiennych globalnych, które zadeklarujemy na samej górze naszej aplikacji. Spójrz na szablon:

```
program P060;{ten program się nie skompiluje)
var zmienna_globalna : typ;

procedurę nazwa(zmienna_lokalna : typ);
var zmienna_lokalna : typ;
begin
   {tutaj widać zmienne lokalne i zmienne globalne}
end;

{program główny}
begin
   {tutaj widać tylko zmienne globalne}
end.
```

Kiedy na samym początku naszej przygody z Pascalem omawialiśmy identyfikatory, wspomnieliśmy, że nie mogą wystąpić w programie dwa identyfikatory o takiej samej treści. W rzeczywistości - nie możemy po prostu dopuścić do sytuacji, w której kompilator nie wiedziałby co uczynić. Może się więc zdarzyć, że w programie wystąpią dwa identyfikatory tak samo brzmiące. Spójrz na przykład:

```
program PO 61;
var x : Integer;

procedure test;
var x : integer;
begin
   X:=5;
end;

{program główny}
begin
   X:=10;
   Test;
   Writeln(x);
   ReadLn;
end.
```

W powyższym programie utworzona została zmienna globalna o nazwie x. W programie głównym, w pierwszej kolejności nadaliśmy jej wartość 10. Następnie wykonaliśmy procedurę Test i po niej wyświetliliśmy wartość tej zmiennej. Jak się okaże - będzie to rzeczywiście liczba 10. Pomimo, iż w procedurze Test nadajemy zmiennej X wartość 5. Jednak w tym podprogramie zadeklarowaliśmy zmienną lokalną o nazwie X. Są więc w programie dwie zmienne o tym samym identyfikatorze. Jednak zmienna lokalna „pokrywa" zmienną globalną wewnątrz podprogramu i to na niej dokonana została zmiana, a nie na zmiennej globalnej. Gdybyśmy chcieli zmienić wartość zmiennej globalnej, wystarczyłoby usunąć deklarację zmiennej X wewnątrz procedury - mogłaby ona wyglądać tak:

```
procedure test;
begin
   X:=5;
end;
```

Teraz nastąpiłaby zmiana zmiennej globalnej.

Spójrz na nasz następny program:

```
program PO62;

procedure Ala(x : Integer);
begin
   X:=10;
end;

procedure Jola;
var X : Integer;
begin
  X:=5;
  Ala(x);
  Writeln(x);
end;

begin
    Jola;
end.
```

Wcześniej zobaczyłeś, w jaki sposób wewnątrz pewnej procedury zmienić wartość zmiennej globalnej. Sprawa jest prosta - wystarczy użyć jej identyfikatora, pod warunkiem, że wewnątrz danego podprogramu nie ma żadnych deklaracji o tej samej nazwie. Teraz problem mamy podobny, ale tamto rozwiązanie nie zadziała.

Mamy dwie procedury Ala i Jola. W programie głównym wywołujemy tą drugą. W podprogramie Jola stworzyliśmy zmienną lokalną o nazwie X. Nadaliśmy jej wartość 5. Ta zmienna widoczna jest tylko w obszarze tej procedury, jako że jest zmienną lokalną. Nigdzie indziej jej nie widać. Chcemy zmienić jej wartość poprzez procedurę Ala. Wywołujemy więc ją z argumentem X. Procedura Ala oczekuje argumentu X typu Integer i nadaje mu wartość 10. Jak się jednak okaże (po uruchomieniu programu) wartość zmiennej X w procedurze Jola nie ulegnie zmianie. Wyświetli się liczba 5. Dlaczego? Bowiem w procedurze Ala tworzona jest również zmienna lokalna o nazwie X. Jest to jednak zupełnie inna zmienna niż w procedurze Jola. Co prawda, przy wywołaniu jej, nadana zostanie zmiennej wartość 5, jednak zapis x:=10 sprawi, że zmienna lokalna X w procedurze Ala przyjmie tę wartość, jednak zmienna lokalna X z procedury Jola się nie zmieni. Cóż więc możemy zrobić? Skorzystać z tzw. referencji. Wystarczy poprzedzić deklarację zmiennej X w nagłówku procedury Ala słowem var i problem się rozwiąże. Oto jak miałaby ona wyglądać:

```
procedure Ala(var x : Integer);
begin
   X:=10;
end;
```

W ten sposób, wywołując tę procedurę nie jest tworzona zmienna lokalna. Tworzony jest jedynie alias do zmiennej przekazanej jako argument, a więc jakby kolejna nazwa w innym bloku. Mogłoby być nawet :

```
procedure Ala(var arg : Integer);
```

I wówczas zmienna arg i zmienna x byłyby tą samą nazwą z procedury Jola (w poprzednim przypadku).

8.4. Rekurencja

Kiedy już nauczyliśmy się tworzyć własne procedury i funkcje, wykorzystywać je w programie głównym lub w innych podprogramach, uważny Czytelnik mógłby zadać pytanie - a co się stanie, gdy wewnątrz pewnego podprogramu wywołamy go samego? Jest to oczywiście możliwe i czasami znacznie upraszcza program. Takie wywołanie samego siebie, a więc inaczej odwoływanie się do siebie samego nazywamy rekurencją. Spójrz na przykład funkcji - klasycznej, rekurencyjnej funkcji silnia.

```
function silnia( x : Byte) : Word;
begin
   if x=0 then silnia:=1
          else silnia:=x*silnia(x-1);
end;
```

Aby wyjaśnić powyższą funkcję trzeba na chwilkę zająć się podstawami matematyki. Jak obliczamy wartość funkcji silnia? Spójrz...

```
0!=1
1!=1
2!=1*2=2
3!=1*2*3=6
4!=1*2*3*4=24
itd.
```

Jak widzisz, silnia z 4 jest równa 4 razy silnia z 3, prawda? Bo to jest 1*2*3 (czyli silnia z 3) * 4. Możemy więc powiedzieć, że silnia z X to X razy silnia z X-1. I to jest zapis rekurencyjny funkcji silnia. Aby jednak program nie zapętlił się w nieskończoność, tzn. nie wywoływał nieskończenie długo samego siebie (bo spowodowałoby to wystąpienie błędu), trzeba jakoś określić, kiedy to wywoływanie ma się skończyć, czyli określić pewien warunek końcowy, tzw. warunek stopu. W przypadku funkcji silnia jest to argument 0. Silnia z 0 wynosi 1 i nie korzystamy

już z rekurencji. Spójrz na przebieg zadziałania programu z argumentem `silnia(3)`.

```
Silnia(3)=3*Silnia(2)
Silnia(2)=2*Silnia(1)
Silnia(1)=1*Silnia(0)
Silnia(0)=1
Więc Silnia(1)=1*1=1
Więc Silnia(2)=2*1=2
Więc Silnia(3)=3*2=6
A więc Silnia(3)=6
```

Rekurencja jest zagadnieniem bardziej zaawansowanym i nie będziemy poświęcać mu więcej uwagi. W rękach programisty jest to jednak potężne narzędzie. Bardzo często upraszczające zapis programu i sprawiające, że jest on znacznie czytelniejszy i bardziej spójny. Często jednak utrudnia jego zrozumienie.

9. Modułowa postać programu

Programy pisane w Turbo Pascalu posiadają tzw. postać modułową a więc składają się z wielu modułów, które w pewien sposób ze sobą współpracują. Korzystałeś już z modułów wielokrotnie, jednak robiłeś to tylko nieświadomie. W tym momencie dowiesz się dokładniej czym one są, kiedy z nich korzystałeś, jakie gotowe moduły dostarcza Turbo Pascal, jak tworzyć i korzystać z nich samodzielnie.

9.1. Moduły

Tak jak procedura, czy funkcja jest pewnym zwartym podprogramem, który wykonuje określone działanie, tak moduł jest pewnym plikiem, który zawiera stałą ilość procedur i funkcji i -jako całość - wykonuje pewne działanie, albo po prosto służy dostarczaniu programiście takich zgrupowanych podprogramów. W swoich programach korzystałeś wielokrotnie z procedur Write, Writeln, Read, ReadLn, itd. Wiesz już dobrze czym są procedury, a czym funkcje. Wiesz, jaką mają budowę. Ale aby skorzystać z własnej procedury, musiałeś ją stworzyć. Procedury wymienione powyżej też zostały stworzone (przez autorów Turbo Pascala) i też mają taką samą budowę. Ty ich jednak nie widzisz, bowiem zostały one wszystkie zapisane w tzw. module System, który dołączany jest do Twojego programu automatycznie. Turbo Pascal udostępnia jednak znacznie więcej procedur gotowych do wykorzystania. Na przykład, procedury obsługujące tryb tekstowy (ustawianie kursora, zmiana koloru liter, zmiana koloru tła), wszystkie razem zostały zgrupowane w module nazwanym Crt. Inne procedury i funkcje umożliwiające przełączenie się do tzw. trybu graficznego, rysowanie okręgów, linii, prostokątów, wypełnianie obszarów, zostały zebrane w module Graph. Są jeszcze moduły Dos, Printer i wiele innych. Jednak tylko System dołączany jest do programu automatycznie i dlatego możemy wyświetlać napisy za pomocą Write, czy Writeln niczego więcej nie określając. Chcąc skorzystać z procedur zapisanych w innych modułach, musimy powiedzieć kompilatorowi, gdzie powinien zajrzeć. Dokonujemy tego jawnie, słowem kluczowym uses występującym na samym początku programu (po nagłówku).

Oto przykład:

```
program PO 63;
uses  Crt;    {deklaracja  modułu}
begin
   GotoXY(5,5);
   Writeln('Napis');
end.
```

W powyższym programie zadeklarowaliśmy moduł Crt, tzn. powiedzieliśmy kompilatorowi, aby tam zajrzał. W tym module zapisana jest procedura GotoXY, która pozwala na ustawianie kursora na ekranie monitora. Gdybyśmy modułu nie zadeklarowali, kompilator nie wiedziałby, czym jest identyfikator GotoXY i nie moglibyśmy z niego korzystać.

9.2. Moduł CRT

Moduł Crt dostarcza programiście wiele procedur i funkcji, usprawniających obsługę trybu tekstowego. Teraz przedstawimy kilka najciekawszych z nich, aby Twoje programy mogły być atrakcyjniejsze i bardziej przyjazne użytkownikowi. Poznałeś już procedurę GotoXY. Oto jej nagłówek:

```
procedure GotoXY(X,Y : Integer);
```

Gdy widzisz taki nagłówek, możesz z niego odczytać czy dany podprogram jest procedurą, czy funkcją. Jakich i ile oczekuje argumentów. Wielokrotnie w tej książce, wprowadzając nowe instrukcje przedstawiamy je właśnie w taki sposób.

Procedura GotoXY służy do zmiany położenia kursora, a więc miejsca w którym wyświetlane są napisy i wpisywane znaki z klawiatury. X to numer kolumny (od 1 do 80), a Y numer wiersza (od 1 do 25). Z instrukcji tej korzystaliśmy już wielokrotnie, dlatego nie potrzeba chyba więcej komentarzy.

```
procedure TextColor(Kolor : Integer);
```

Procedura TextColor umożliwia zmianę koloru wyświetlanych napisów. Jednak zmiana nastąpi w dopiero wyświetlanych tekstach, a nie w tych już wyświetlonych. Jako argument podajemy numer koloru. Oto pomocna tabelka:

Numer koloru	Stała	Kolor
0	Black	Czarny
1	Blue	Niebieski
2	Green	Zielony
3	Cyan	Morski

4	Red	Czerwony
5	Magenta	Fioletowy
6	Brown	Brązowy
7	White	Biały
8	Gray	Szary
9	Light blue	Jasnoniebieski
10	Light green	Jasnozielony
11	Light cyan	Jasny morski
12	Light red	Jasnoczerwony
13	Light magenta	Jasnofioletowy
14	Yellow	Żółty
15	Bright white	Rozjaśniony biały

Możemy posłużyć się liczbowym numerem koloru i możemy również wykorzystać stałą, która jest bardziej intuicyjna i która jest równa odpowiedniej liczbie.

```
procedure TextBackGround(Kolor : Integer);
```

Powyższa procedura zmienia kolor tła wyświetlanego tekstu. Podobnie jak TextColor, oczekuje liczbowego numeru koloru. Jednak obsługiwanych jest tylko pierwszych 8 kolorów z tabeli powyżej.

Oto prosty przykład pokazujący wykorzystanie kolorów w Turbo Pascalu.

```
program PO 64;
uses Crt;
var kolor,tlo,x : Integer;
begin
   for kolor:=0 to 15 do
      for tlo:=0 to 7 do
      begin
          X:=WhereX;

          TextColor(kolor);
          TextBackGround(tlo);

          Writeln(kolor:2,'-',tlo,' Pascal ');

          GotoXY(X,WhereY);
          if WhereY>=24 then GotoXY(WhereX+13,1);

      end;
    Readln;
end.
```

W powyższym programie skorzystaliśmy z dwóch nowych funkcji, mianowicie WhereX i WhereY. Tak jak instrukcja GotoXY ustawia kursor w wybranym miejscu na ekranie, tak te dwie funkcje zwracają aktualną współrzędną kursora. Spróbuj samodzielnie przeanalizować program traktując ekran wynikowy jako podpowiedz. Nie powinieneś mieć z tym większych problemów.

```
procedure ClrScr;
```

Instrukcja powoduje wyczyszczenie ekranu. Jeśli, przykładowo, kolor tła ustawiony jest na kolor czerwony - tło zostanie na taki kolor wyczyszczone. Możesz każdy swój program poprzedzać tą instrukcją, aby wyczyścić wszystkie występujące aktualnie na nim napisy.

Spójrz na przykład:

```
program PO 65;
uses Crt;
var i : Integer;
begin
   for i:=0 to 7 do
   begin
     TextBackGround(i);
     ClrScr;
     ReadLn;
   end;
end.
```

Po uruchomieniu tego programu ekran będzie zmieniał kolor w tempie wciskania przez Ciebie klawisza [ENTER].

```
procedure Sound(Hz : Integer);
```

Procedura powoduje wygenerowanie dźwięku o wskazanej częstotliwości. Dźwięk będzie wydobywał się tak długo, aż zostanie wyłączony poniższą procedurą:

```
procedure NoSound;
```

Bardzo często wykorzystywaną instrukcją jest:

```
procedure Delay(S : Integer);
```

Powoduje ona zatrzymanie wykonywania programu na określoną liczbę milisekund. Korzystaliśmy z niej już w kilku programach, dlatego nie wymaga ona dodatkowych komentarzy.

Spójrz teraz na poniższy program:

```
program PO 66;
uses Crt;
var Klawisz : Char;
procedurę Graj(Hz,S : Integer);
begin
  Writeln('Dźwięk: ',Hz,'Hz, trwający ',S,
          ' milisekund.');
  Sound(Hz);
  Delay(S);
  NoSound;
end;
begin
   repeat
     if KeyPressed Then Klawisz:=UpCase(ReadKey)
                   else Klawisz:=#0;
     case Klawisz of
     'Q'  | Graj|100 100)
     'W   | Graj|200 100)
     'E'  | Graj|300 100)
     'R'  | Graj|400 100)
     'T'  | Graj|500 100)
     'Y'  | Graj|600 100)
     'U'  | Graj|700 100)
     'I'  | Graj|800 100)
     'A'  | Graj|100 200)
     'S'  | Graj|200 200)
     'D'  | Graj|300 200)
     'F'  | Graj|400 200)
     'G'  | Graj|500 200)
     'H'  | Graj|600 200)
     'J'  | Graj|700 200)
     'K'  | Graj|800 200)
     'Z'  | Graj|100 300)
     'X'  | Graj|200 300)
     'C'  | Graj|300 300)
     'V'  | Graj|400 300)
     'B'  | Graj|500 300)
     'N'  | Graj|600 300)
     'M'  | Graj|700 300)
     ' '  | Graj|800 300)
   ena;
   until Klawisz=#27;
end.
```

Program przedstawiony przed chwilą, to proste pianinko. Wciskając różne klawisze na klawiaturze, spowodujemy, że z głośniczka dochodzić będą odpowiednie dźwięki.

W powyższym programie skorzystaliśmy z nowej funkcji ReadKey. Jeśli użytkownik wciska na klawiaturze jakiś klawisz, funkcja KeyPressed zwraca wówczas wartość TRUE, co oznacza „wciśnięto przycisk". W tym momencie funkcja ReadKey zwróci znak, jaki został wciśnięty. Ponieważ chcieliśmy, aby w naszym programie nie było ważne, czy ktoś wciska małe literki, czy też wielkie, każdy wciskany znak zamieniamy na wielki (UpCase) i porównuje na liście ze znakami również jako wielkie przedstawionymi.

Pętla w programie wykonuje się tak długo, aż użytkownik wciśnie klawisz, który ma kod ASCII 27. Jest nim klawisz [ESC].

9.3. Własny moduł

W poprzednim punkcie napisaliśmy wspólnie proste pianinko. W innym programie stworzyliśmy odrębną procedurę, która wydobywała różne dźwięki. Kiedy w przyszłości będziesz pisał podobny program, w którym chciałbyś wykorzystać tą procedurę, będziesz musiał ją ponownie przepisywać. Możesz jednak stworzyć własny moduł, w którym ją umieścisz, oraz inne, najlepsze i najfajniejsze własne procedury oraz funkcje. W przyszłości, klauzulą uses, będziesz mógł dołączyć ten moduł do każdego swojego programu i korzystać z gotowych podprogramów. W ten sposób Twój kod źródłowy programu nie będzie się rozrastał. Bardzo zachęcam do takiego podejścia przy programowaniu.

Każdy moduł w Turbo Pascalu posiada specyficzną budowę.

Oto jego szablon:

```
unit nazwa;

interface
{blok z deklaracjami}

imp1ementation
{blok implementacyjny}

begxn
  {kod inicjujący}
end.
```

Każdy moduł rozpoczyna się słowem unit, po którym następuje nazwa modułu. Pod taką samą nazwą moduł musi być zapisany na dysku (nie było takiej konieczności w przypadku nagłówka program). Każdy moduł składa się z dwóch głównych bloków. Bloku interface i bloku implementation. Może również zawierać blok inicjujący umieszczony na końcu (begin...end.). Jeśli nie posiada tego bloku, i tak musi być zakończony słowem end. (bez słowa begin).

Kiedy dołączamy moduł do programu, poprzez słowo kluczowe uses następuje uruchomienie bloku inicjującego (jeśli zostanie zdefiniowany). Dlatego instrukcje, które w nim umieścisz, wykonają się na samym początku programu (jeszcze przed instrukcjami z programu głównego).

W części implementation umieszczasz wszystkie procedury i funkcje, jakie są niezbędne w danym module. Natomiast w części interface umieszczasz tylko nagłówki tych z nich, które chcesz, aby mogły być wykorzystywane w programie, do którego dany moduł jest dołączony (jest to więc pewien sposób hermetyzacji). Oto przykład modułu:

```
unit PO 67;

interface
uses Crt;

   procedure Wyświetl(X,Y : Integer;Napis : String);
   procedure Graj(Hz,S : Integer);
implementation
   procedure Graj(Hz,S : Integer);
   begin
     Sound(Hz);
     Delay(S) ,-
     NoSound;
   end;
   procedure Wyświetl(X,Y : Integer; Napis : String);
   begin
       GotoXY(X,Y);
       Write(Napis);
   end;

end.
```

Powyższy moduł nosi nazwę PO67. Nie zawiera części inicjującej, dlatego kończy się słowem end. i nie posiada słowa begin. Ponieważ korzystamy w nim z instrukcji należących do modułu Crt, dokonaliśmy jego deklaracji jak w zwykłym programie, jednak po słowie interface. Moduł zawiera dwie procedury, których „ciało" umieściliśmy w bloku implementation. Chcemy, aby oba z

tych podprogramów mogły być wykorzystywane przez programistów w swoich aplikacjach, dlatego ich nagłówki umieściliśmy jeszcze w części interface. Teraz można napisać taki program:

```
program PO68;
uses PO67;
begin
    Wyswietl(10,10,'Turbo  Pascal');
    Graj(500,500);
    ReadLn;
end.
```

10. Obsługa plików

W tym rozdziale znalazły się informacje na temat obsługi plików z poziomu Turbo Pascala. Dowiemy się, w jaki sposób tworzyć, odczytywać i zapisywać własne dane do pliku. Dzięki temu będziemy w stanie stworzyć bardziej zaawansowane programy, własne książki telefoniczne, słowniki, czy inne proste bazy danych.

10.1. Skojarzenie plików

Podczas operowania na plikach, a to oznacza: podczas zapisywania w nich pewnych informacji i czytania ich, nie posługujemy się nazwami plików. Byłoby to bardzo uciążliwe. Posługujemy się natomiast specjalnymi zmiennymi, tzw. zmiennymi plikowymi. Przed otwarciem zbioru, czy zapisem do niego danych - taką zmienną plikową musimy skojarzyć z plikiem na dysku. Wykorzystujemy w tym celu instrukcję Assign. Tylko wówczas podajemy jego pełną ścieżkę i nazwę pliku. Oto przykład:

```
var Plik : Text;

   Assign(Plik,'c:\plik.txt');
```

W ten sposób zadeklarowaliśmy zmienną plikową Plik, którą poprzez instrukcję Assign skojarzyliśmy ze zbiorem c:\plik.txt. Ponieważ zmienna jest typu Text, plik którym będziemy operowali powinien być dokumentem tekstowym.

Skojarzenie zmiennej z plikiem nie wiąże się z jego otwarciem, ani nawet jakąkolwiek próbą jego modyfikacji. Dlatego plik podany jako argument do procedury Assign może w ogóle nie istnieć na dysku. Może to być zbiór, który chcemy dopiero stworzyć.

Istnieją jeszcze inne typy plikowe. Przykładowo:

```
var Plik : file of Byte;
```

Zmienne tego typu powinny zawierać uchwyty do tzw. plików binarnych. Do takich zbiorów będziemy mogli zapisywać i odczytywać z nich dane pojedynczymi bajtami (w plikach tekstowych - liniami tekstu). Po słowach kluczowych file of możemy wstawić dowolny typ danych (również rekordowy), np.:

```
var Plik : file of Word;
```

Pliki skojarzone z taką zmienną będą mogły być czytane i zapisywane **całymi** słowami, tzn. większymi liczbami całkowitymi.

10.2. Zapisywanie do pliku

Aby zapisać jakieś informacje do pliku trzeba go najpierw stworzyć (jeśli **nie** istnieje) lub otworzyć i usunąć jego zawartość. Obie te czynności wykonuje **jedna** instrukcja, mianowicie Rewrite. Jako argument oczekuje zmiennej plikowej, która uprzednio została skojarzona z określonym plikiem. Po zapisaniu wszystkich niezbędnych informacji, każdy dokument trzeba zamknąć. Służy do tego **celu** instrukcja Close. Szablon postępowania wygląda następująco.

```
var Plik : typ_plikowy;

Assign(Plik,nazwa_pliku);
Rewrite(Plik);

Close(Plik);
```

Jeśli nie zamkniemy na koniec pliku, może się okazać, że nie zawiera on **danych,** które sądziliśmy, że zostały zapisane. Bowiem wszystkie informacje, **które** wysyłamy do zbioru, są buforowane (umieszczane w pamięci) i przenoszone fizycznie do pliku, dopiero przy jego zamknięciu.

Do zapisywania danych w pliku korzystamy z dwóch, poznanych już wcześniej. procedur. Są to Write i Writeln. Wcześniej uczyłeś się, jak z nich korzystać w przypadku wyświetlania napisów. W identyczny sposób wykorzystuje się je **przy** zapisywaniu danych do pliku. Różnica polega na argumentach. Jako pierwszy argument, w tym przypadku, trzeba podać zmienną plikową wskazującą **plik** uprzednio otwarty. Oto przykład:

```
program PO_6_9;
var Plik : Text;
begin
   Assign(Plik,'c:\plik.txt');
   Rewrite(Plik);
   Writeln(Plik,'Pierwsza linijka pliku');
   Write(Plik, 'Druga ');
   Write(Plik,'linijka     ');
   Writeln(Plik,'pliku.');
   Writeln(Plik,'Trzecia linijka pliku');
   Close(Plik);
end.
```

Podobnie jak w przypadku wyświetlania napisów, instrukcja Writeln po zapisaniu przekazanego jako argument tekstu w pliku, dostawia za nim sekwencję nowej linii tak, że przy jego odczycie dane są umieszczone jedna pod drugą.

Przy użyciu instrukcje Writeln można zapisywać informacje tylko do plików skojarzonych ze zmienną typu Text. Bowiem w przypadku plików binarnych nie można zapisywać danych liniami. W takim wypadku zapisuje się dane całymi rekordami (dosłownie - dane typu rekordowego lub po prostu bajtami, słowami, danymi typu Real, itp.) Korzystamy wówczas z instrukcji Write. Oto przykład.

```
program P070;
var Plik : file of Byte;
    B    : Integer;
begin
   Assign(Plik,'c:\plik.dat');
   Rewrite(Plik);
   B:=23;
   Write(Plik,B);
   B:=12;
   Write(Plik,B);
   Close(Plik);
end.
```

10.3. Dopisywanie do pliku

Gdy otwieramy istniejący plik instrukcją Rewrite jego zawartość jest usuwana. Czasami jednak zachodzi potrzeba dopisania pewnych informacji do pliku już istniejącego. Jednym z rozwiązań jest wówczas jego przeczytanie i zapamiętanie całej zawartości, a następnie otworzenie do zapisu i wpisanie wcześniej zapamiętanych danych, dopisując na końcu te nowe. Nie jest to jednak rozwiązanie optymalne, bo Turbo Pascal pozwala na otworzenie pliku w tzw. trybie do dopisu. Aby tego dokonać, zamiast Rewrite, wykorzystujemy Append. Reszta - a więc zapisywanie danych i zamknięcie zbioru -jest identyczna. Oto przykład:

```
program P071;
var Plik : Text;
begin
   Assign(Plik,'c:\plik.txt');
   Append(Plik);
   Writeln(Plik,'Czwarta linia');
   Close(Plik);
end.
```

Do dopisu można jednak otworzyć plik, który już istnieje. W jaki bowiem sposób dopisać dane do zbioru nieistniejącego? Taka próba zakończy się niepowodzeniem.

10.4. Odczytywanie z pliku

Nauczyliśmy się zapisywać i dopisywać dane do zbiorów, pora nauczyć się je czytać. Aby odczytać plik trzeba go otworzyć w trybie do odczytu. Dokonujemy tego instrukcją Reset. Następnie - poprzez znane Ci procedury Read i ReadLn - czytamy dane. Na koniec, oczywiście, zamykamy dokument.

Instrukcje Read i ReadLn pobierały dotychczas dane od użytkownika. Podobnie jak Write i Writeln -podając jako pierwszy argument zmienną plikową - będą pobierać dane nie z klawiatury, lecz z pliku.

Instrukcja ReadLn wczytuje całe linie tekstu i może być wykorzystywana tylko przy plikach tekstowych (Text). Read może pobierać pojedyncze litery (pliki tekstowe), jak również dowolne inne rekordy.

Abyśmy mogli otworzyć plik do odczytu musi on oczywiście istnieć. Nie można doprowadzić do sytuacji, że następuje próba odczytania dokumentu, który nie istnieje. Aby sprawdzić, czy wskazany zbiór istnieje można posłużyć się funkcją FSearch. Oto przykład.

```
program PO7_2;
uses Dos;
var Plik : Text;
    Linia: String;
begin
   if FSearch('c:\plik.txt','')='' then
   begin
     Writeln('Brak pliku.');
   end else
   begin
     Assign(Plik,'c:\plik.txt');
     Reset(Plik);
     ReadLn(Plik,Linia);
     Close(Plik);

     Writeln(Linia);
   end;
   ReadLn;
end.
```

Ponieważ funkcja FSearch znajduje się w module Dos, umieściliśmy go na liście deklarującej moduły (uses). Funkcja ta oczekuje dwóch argumentów. Pierwszym z nich jest plik, który ma być odszukany, natomiast drugi to lista katalogów, które mają być przeszukiwane. W naszym przykładzie nie określamy listy katalogów, a więc szukany jest zbiór tylko w katalogu aktualnym (lub wskazanym w ścieżce nazwy pliku). W przypadku odnalezienia zbioru (a więc kiedy on istnieje) funkcja zwraca jego adres. Gdy plik nie zostanie odnaleziony funkcja zwróci łańcuch pusty.

Często zdarza się, że konieczne jest przeczytanie wszystkich danych w takim pliku nie znając informacji na temat ich ilości. Przykładowo - chcemy wyświetlić na ekranie zawartość zbioru C:\AUTOEXEC.BAT. Nie wiemy w danej chwili, ile linii ten plik zajmuje i ile razy powinniśmy wykonać instrukcję ReadLn. Jeśli wykonamy za dużo - wystąpi błąd. Jak więc tego dokonać? Dowiemy się już za chwilę.

Wewnątrz każdego otwartego pliku występuje tzw. wskaźnik. Kiedy z niego czytamy dane, przesuwa się on coraz dalej. Czytając bajtami - przesuwa się za każdym razem o jeden bajt, a czytając liniami - co linię. To właśnie on pokazuje, co dana instrukcja ma odczytać. W Turbo Pascalu występuje funkcja Eof, która zwraca TRUE, kiedy ów wskaźnik znajdzie się na samym końcu otwartego pliku. To świadczy o tym, że nie możemy już z niego nic przeczytać. Możemy wiec skorzystać z tej funkcji pisząc naszą aplikację. Oto przykład:

```
program PO73;
var Plik : Text;
    Linia: String;
begin
   Assign(Plik,'c:\autoexec.bat');
   Reset(Plik);
   while not eof(Plik) do
   begin
      ReadLn(Plik,Linia);
      Writeln(Linia);
   end;
   Close(Plik);
   ReadLn;
end.
```

10.5. Wskaźnik w pliku

W poprzednim punkcie wspomniałem o istnieniu specjalnego wskaźnika w każdym otwartym pliku, który wskazuje, co instrukcje Read i ReadLn mają czytać. Podczas kolejnych odczytań wskaźnik ten odpowiednio się przesuwa, osiągając w pewnym momencie koniec zbioru. Turbo Pascal udostępnia instrukcję Seek, która pozwala na „ręczne" przemieszczanie owego wskaźnika. Dzięki temu możemy dokładnie wskazać miejsce w pliku, pod którym chcemy zapisać pewne dane lub, z którego te dane chcemy odczytać. Procedura Seek oczekuje dwóch argumentów. Pierwszym z nich jest zmienna plikowa, natomiast drugim numer bajta w pliku, pod który wskaźnik ma zostać przesunięty. Oto przykład:

```
program P074;
var Plik : file of Byte;
    Bajt : Byte;
    Wielkosc : Longint;
begin
    Assign(Plik,'c:\plik.dat');
    Reset(Plik);
    Wielkosc:=FileSize(Plik);
    Seek(Plik,l);
    Read(Plik,Bajt);
    Close(Plik);
    Writeln('Plik zajmuje ',Wielkość,' bajtów.');
    Writeln('Pod drugim bajtem zapisana jest <
             liczba: ',Bajt);
    Readln;
end.
```

W powyższym programie pokazane zostało również jak sprawdzić wielkość otwartego pliku. Wystarczy, bowiem skorzystać z funkcji FileSize.

Funkcje Seek i FileSize można wykorzystywać tylko w przypadku plików binarnych.

11. Tryb graficzny

Przyszła kolej na jeden z najprzyjemniejszych aspektów programowania w Turbo Pascalu, mianowicie programowanie w tzw. trybie graficznym.

Dotychczas, wszystkie nasze programy działały w trybie tekstowym. Oznacza to, że ekran podzielony był na 25 wierszy i 80 kolumn. W każdym jego punkcie (określonym pozycją kursora) mogliśmy wyświetlić jeden, dowolny znak. Nie można jednak było rysować okręgów, linii, czy innych figur geometrycznych, chyba że za pomocą gwiazdek albo innych znaków. Kiedy włączy się tzw. tryb graficzny ekran będzie zorganizowany inaczej. Może posiadać, np. 640 kolumn i 480 wierszy, a więc łącznie dużo więcej „punktów" niż w trybie tekstowym. Będą one jednak znacznie mniejsze - bowiem wielkość ekranu się nie zmienia. W trybie graficznym możemy określić kolor każdego z tych punktów. Turbo Pascal oferuje również szereg procedur, które potrafią rysować okręgi, linie, prostokąty, itp. W tym rozdziale zajmiemy się nimi bliżej.

11.1. Inicjowanie trybu graficznego

Jak już zaznaczyliśmy na wstępie, aby móc rysować dowolne figury geometryczne trzeba przełączyć się w tzw. tryb graficzny. Służy do tego celu specjalna procedura `InitGraph`. Jej definicja, jak również definicje wszystkich innych procedur graficznych, zebrane zostały w module `Graph`. Stąd też, w każdym z naszych programów będziemy musieli go zadeklarować.

Istnieje wiele modeli kart graficznych, czyli urządzeń odpowiedzialnych za wyświetlanie grafiki na ekranie monitora. Różnią się one miedzy sobą, jedne mniej, inne bardziej. Są to różnice natury technicznej, które jednak sprawiają, że w trochę inny sposób trzeba „rozmawiać" z poszczególnymi typami kart graficznych, aby narysować linię. My tym oczywiście nie musimy się przejmować, bowiem procedury z modułu `Graph` wykonują to wszystko za nas. Jednak przy włączaniu trybu graficznego musimy poinformować kompilator, z jakiej karty graficznej korzystamy (wtedy będzie potrafił się z nią porozumieć i wykorzystać te biblioteki funkcji, które obsługują naszą kartę). Oprócz tego, że są różne modele kart graficznych, są też różne tzw. tryby graficzne. Można, bowiem przełączyć się do trybu z rozdzielczością 640x480 (szerokość i wysokość ekranu) z 16 kolorami, a można również przełączyć się do trybu 800x600, czy jeszcze innego. Taki również

parametr musimy przekazać procedurze InitGraph. Jednak skąd biedny programista miałby wiedzieć, jaką kartę graficzną posiada użytkownik jego programu? Jaki jest „najlepszy" tryb graficzny? Turbo Pascal został wyposażony w jeszcze inną procedurę o nazwie DetectGraph, która te informacje uzyska. Spójrz na przykład:

```
program P075;
uses Graph;          {deklaracja modułu Graph}
var Karta,Tryb : Integer;
begin
   {rozpoznajemy kartę graficzną i tryb}
   DetectGraph(Karta,Tryb);
   {włączamy tryb graficzny}
   InitGraph(Karta,Tryb,'c:\bp\bgi');
   Circle(50,50,20);      {rysujemy okrąg}
   ReadLn;
   CloseGraph;          {wyłączamy tryb graficzny}
end.
```

Przy instalacji Turbo Pascala, w jego katalogu głównym, utworzony został podkatalog o nazwie bgi. Znajdują się w nim biblioteki graficzne, obsługujące różne karty graficzne. Są to pliki z rozszerzeniem bgi.

Powyższy program rozpoczyna się procedurą DetectGraph, która rozpoznaje, jaki model karty graficznej posiadamy (a dokładniej, której biblioteki użyć) i jaki jest dla niej najlepszy tryb graficzny. Obie te wartości zapamiętuje w dwóch zmiennych przekazanych jako argument. Są to liczby całkowite (Integer), które te wiadomości reprezentują. Następnie wywołujemy procedurę InitGraph, aby włączyć opisywany tryb. W tym celu - jako argumenty - przekazujemy numery karty graficznej i trybu, jaki chcemy zainicjować. W naszym przykładzie wykorzystujemy zmienne, pod które poprzednia procedura zapamiętała prawidłowe wartości. Trzeci argument instrukcji InitGraph to katalog, w którym mają być szukane biblioteki obsługujące poszczególne tryby i karty. Kolejna linia programu to wywołanie instrukcji Circle. Służy ona do rysowania okręgów. Przekazujemy jej trzy argumenty: współrzędne X i Y środka oraz promień R. Na koniec programu zamykamy tryb graficzny, tym samym włączając z powrotem tryb tekstowy. Dokonujemy tego instrukcją CloseGraph.

11.2. Podstawowe instrukcje graficzne

Kiedy potrafimy zainicjować (włączyć) tryb graficzny, jedyne czego musimy jeszcze się nauczyć, to podstawowe instrukcje, które umożliwiają rysowanie w tym trybie. Gdy tego dokonamy, już tylko nasza fantazja i twórcza myśl może być pewnym ograniczeniem. Jednak praktyka czyni mistrza.

11.2.1. Rysowanie linii i okręgów

Okręgi nauczyliśmy się rysować już na samy wstępie, kiedy poznaliśmy metodę włączania trybu graficznego. Aby narysować okrąg, wystarczy skorzystać z instrukcji Circle. Oto jej nagłówek:

```
procedure Circle(X,Y,R : Integer);
```

Jak widzimy, aby z niej skorzystać należy określić trzy argumenty. Pierwsze dwa, to współrzędne środka na ekranie monitora, natomiast ostatni argument określa promień okręgu.

Załóżmy, że chcemy narysować okrąg na środku ekranu. Jakie powinniśmy wprowadzić współrzędne? X powinno wynosić połowę szerokości ekranu, natomiast Y połowę wysokości. Pytanie tylko, jak szeroki i jak wysoki jest ekran w danym trybie, przy danej karcie graficznej? Turbo Pascal udostępnia dwie funkcje, mianowicie GetMaxX i GetMaxY, które te wartości zwracają. Nic już nie stoi na przeszkodzie, aby nasz okrąg narysować.

```
program PO 7 6;
uses Graph;
var Karta,Tryb : Integer;
begin
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');
   Circle(GetMaxX div 2, GetMaxY div 2, 20);
   ReadLn;
   CloseGraph;
end.
```

Do rysowania dowolnych linii (odcinków) możemy wykorzystać instrukcję Line. Oto jej nagłówek:

```
procedure Line(X1,Y1,X2,Y2 : Integer);
```

Przekazujemy instrukcji cztery argumenty. Są to współrzędne dwóch końców określonego odcinka. Łącząc ze sobą linie, możemy tworzyć bardziej rozbudowane figury geometryczne, np. prostokąty, trójkąty, itp.

Spójrz na kolejny przykład:

```
program P077;
uses Graph;
var Karta,Tryb : Integer;
begin
   DetectGraph(Karta,Tryb) ;
   InitGraph(Karta,Tryb,'c:\bp\bgi');

   SetColor(RED);
   Circle(100,100,50) ;
   SetColor(YELLOW);
   Line(8,150,194,150) ;
   Line(8,150,100,10) ;
   Line(100,10,194,150) ;

   ReadLn;
   CloseGraph;
end.
```

W powyższym przykładzie wykorzystaliśmy nową procedurę SetColor, do ustalenia koloru rysowanych figur. Jej argumentem jest numer koloru. My skorzystaliśmy ze stałej zdefiniowanej w module Graph (stałe i kolory mają te same nazwy i numery, jak w trybie tekstowym)

Przeanalizuj kolejny przykład:

```
program PO 7 8;
uses Crt,Graph;
var Karta,Tryb : Integer;
    X,Y,A,B : Integer;
begin
   Randomize;
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');

     =Random(GetMaxX-10 0)+50;
     =Random(GetMaxY-100)+50 ;
     =1;
     = 1;
```

```
    repeat
      {rysujemy    okrąg}
      SetColor(Yellow);
      Circle(X,Y,10);
      Delay(5);
      {zmazujemy    okrąg}
      SetColor(Black);
      Circle(X,Y,10);
      Inc(X,A);
      Inc(Y,B);
      if (x<10) or (x>GetMaxX-10) then A = -A;
      if (y<10) or (y>GetMaxY-10) then B = -B;
    until KeyPressed;
    CloseGraph;
end.
```

Efektem jest okrąg odbijający się od krawędzi ekranu. Dzięki temu, że po narysowaniu i krótkiej pauzie rysujemy ten sam kształt na kolor czarny (czyli „zmazujemy" go) okrąg nasz nie pozostawia za sobą śladu. Możesz jednak spróbować dokonać pewnych modyfikacji w kodzie, aby ślad był pozostawiany.

Współrzędne środka okręgu cały czas (w pętli) zwiększamy o współczynniki A i B. Przyjmują one wartości 1 i - 1 , w zależności od tego, w którą stronę okrąg się przesuwa. Algorytm jest, więc niemalże identyczny jak ten prezentowany wcześniej, przy okazji poruszania gwiazdki w trybie tekstowym.

Dla przećwiczenia spróbuj dodać jeszcze jeden okrąg poruszający się na ekranie. Następnie współrzędne tych dwóch okręgów połącz odcinkiem. W ten sposób uzyskasz efekt linii odbijającej się od krawędzi ekranu. Kiedy tego dokonasz (również samodzielnie) dodaj jeszcze jedną linię tak, aby poruszały się dwie (w różnych kolorach) niezależnie od siebie.

Spójrz na nasz następny przykład:

```
program PO79;
uses Crt,Graph;
var Karta,Tryb : Integer;
    X,Y,A,B : Integer;
```

```
begin
   Randomize;
   DetectGraph(Karta,Tryb
   InitGraph(Karta,Tryb,'c \bp\bgi'
   X:=Random(GetMaxX div 2--100)+50;
   Y:=Random(GetMaxY div 2--100)+50;
   A:=1;
   B:=1;
   repeat
      {rysujemy okrąg)
      SetColor(Yellow);
      Circle(X,Y,10);
      SetColor(GREEN);
      Circle(GetMaxX-X,Y,10);
      Delay(5);
      {zmażujemy okrąg}
      SetColor(Black);
      Circle(X,Y,10);
      Circle(GetMaxX-X,Y,10);
      Inc(X,A);
      Inc(Y,B);
      if (x<10) or (x>GetMaxX div 2-10) then A:=-A;
      if (y<10) or (y>GetMaxY-10) then B:=-B;
   until KeyPressed;
   CloseGraph;
end.
```

Powyższy program jest bardzo podobny do przedstawionego uprzednio. Zmiana polega na tym, że prawą krawędź, od której ma się odbijać okrąg, przesunęliśmy do środka ekranu. W ten sposób nasza figura porusza się tylko w jednej jego połowie. Innym kolorem rysujemy okrąg symetryczny względem osi OY, a więc porusza się on takim samym ruchem w prawej połówce monitora.

Spróbuj samodzielnie dodać jeszcze jeden okrąg poruszający się w lewej połówce i symetryczny do niego w prawej połówce. Następnie połącz okręgi linią (zarówno te z lewej strony jak i z prawej). Kiedy już to uczynisz, spróbuj ograniczyć ekran poruszania się okręgów, czy linii do jednej ćwiartki ekranu i dodaj symetryczne figury dla każdej pozostałej ćwiartki. W efekcie powinieneś uzyskać efekt poruszających się dwóch okręgów w każdej ćwiartce (łącznie osiem), lub po jednej linii w każdej ćwiartce.

11.2.2. Prostokąty

Aby narysować prostokąt, można skorzystać z procedury Linę i odpowiednio połączyć cztery odcinki. Jednak Turbo Pascal udostępnia gotową instrukcję, realizującą to zadanie. Oto jej nagłówek:

```
procedure Rectangle(X1,Y1,X2,Y2 : Integer);
```

Jako argumenty podajemy współrzędne dwóch punktów, opisujących przekątną danego prostokąta. Oto przykład:

```
program PO 80;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;
begin
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb, 'c:\bp\bgi') ;
   for i:=1 to 15 do
   begin
     SetColor(i);
     Rectangle(i*10,i*10,i*15+200,i*15+200);
   end;
   ReadLn;
   CloseGraph;
end.
```

Prostokąty mogą być również wypełnione. Wówczas korzystamy z analogicznej procedury o nazwie Bar.

11.2.3. Punkty

Wszystkie figury, które rysujemy na ekranie, składają się z punktów - pikseli. Pascal pozwala na rysowanie dowolnych krzywych właśnie za ich pomocą. Oto składnia procedury PutPixel.

```
procedure PutPixel(X,Y,Kolor : Integer);
```

Jako argument przekazujemy trzy liczby. Pierwsze dwie określają współrzędne na ekranie, natomiast trzecia to kolor punktu. Spójrz na poniższy przykład, który rysuje okrąg za pomocą pojedynczych punktów.

```
program P0 81;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;
    X,Y : Integer;
begin
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');
   for i:=1 to 3 60 do
   begin
     x:=Round(Sin(i*pi/180) * 100);
     y:=Round(Cos(i*pi/180) * 100);
     PutPixel(X+GetMaxX div 2,
              Y+GetMaxY div 2,YELLOW);
     Delay(2);
   end;
   ReadLn;
   CloseGraph;
end.
```

Ponieważ przy wyznaczaniu współrzędnych X i Y posługujemy się funkcjami Sin i Cos, które zwracają liczby rzeczywiste, a instrukcja PutPixel oczekuje jako argumentów współrzędnych w postaci liczb całkowitych, posłużyliśmy się funkcją Round, by wartości rzeczywiste zaokrąglić i skonwertować do liczb całkowitych.

Funkcja GetPixel pozwala na sprawdzenie, jakiego koloru jest dany punkt na ekranie. Jako argument oczekuje ona dwóch liczb będących współrzędnymi danego punktu. Funkcja zwraca numer koloru.

Napiszmy teraz prosty program, który będzie potrafił zapisywać do pliku aktualny ekran, a następnie będzie powodował wyczyszczenie jego zawartości i wczytanie uprzednio zapisanego zbioru.

```
program PO 82;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;

procedure ZapiszDoPliku(N : String);
var Plik   file of Byte;
    X,Y    Integer;
    Kolor  Byte;
```

```
begin
   Assign(Plik,N);
   Rewrite(Plik);
   for y:=l to GetMaxY do
     for x:=l to GetMaxX do
     begin
        Kolor:=GetPixel(X,Y);
        Write(Plik,Kolor);
     end;
   Close(Plik);
end;
procedure WczytajZPliku(N : String),-
var Plik : file of Byte;
    X,Y  : Integer;
    Kolor: Byte;
begin
   Assign(Plik,N);
   Reset(Plik);
   for y:=l to GetMaxY do
     for x:=l to GetMaxX do
     begin
       Read(Plik,Kolor);
       PutPixel(x,y,Kolor);
     end;
   Close(Plik);
end;
begin
   Randomize;
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');
   for i:=l to 100 do
   begin
      SetColor(Random(15));
      Line(GetMaxX div 2, GetMaxY div 2, -4
           Random(GetMaxX),Random(GetMaxY));
   end;
   ZapiszDoPliku('c:\rysunek.dat');
   ClearDevice;     {czyszczenie ekranu}
   Delay(lOOO);
   WczytajZPliku('c:\rysunek.dat');
   ReadLn;
   CloseGraph;
end.
```

W powyższym programie skorzystaliśmy z nowej instrukcji ClearDevice w celu wyczyszczenia ekranu. Bowiem w trybie graficznym nie możemy posługiwać się procedurą ClrScr, występującą w trybie tekstowym.

11.3. Wypełnianie obszarów

W poprzednim punkcie nauczyliśmy się jak rysować wypełnione prostokąty. Jednak nie potrafiliśmy nawet zmienić koloru owego wypełnienia. SetColor nie pomagało. Turbo Pascal posiada jednak specjalną instrukcję - SetFillStyle - za pomocą której możemy zmienić nie tylko kolor, ale i sposób wypełniania różnych obszarów. Oto jej składnia:

```
procedure SetFillStyle(Rodzaj, Kolor : Integer);
```

Dostępnych jest 10 różnych rodzajów wypełnienia. Spójrz na przykład:

```
program PO83;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;
begin
   Randomize;
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');
   for i:=1 to 10 do
   begin
       SetFillStyle(i,Random(15)+1);
       Bar(Random(GetMaxX),Random(GetMaxY), M
                  Random(GetMaxX),Random(GetMaxY));
   end;
   ReadLn;
   CloseGraph;
end.
```

Jednak mógłbyś zadać pytanie - a jak narysować wypełniony okrąg? Niestety Turbo Pascal nie posiada specjalnej instrukcji wykonującej to zadanie, ale posiada inną instrukcję, mianowicie FloodFill, dzięki której można sprostać temu problemowi.

Instrukcja FloodFill umożliwia wypełnienie dowolnych obszarów zamkniętych określonym kolorem.

Oto jej składnia:

```
procedure FloodFill(X,Y,Ramka : Integer);
```

Argumentami tej instrukcji są współrzędne dowolnego punktu należącego do danego obszaru zamkniętego i kolor obramowania tego obszaru. Poniżej znajduje się przykładowy program wykorzystujący możliwości owej procedury.

```
program P084;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;

begin
   Randomize;
   DetectGraph(Karta,Tryb);
   InitGraph(Karta,Tryb,'c:\bp\bgi');

   {rysujemy żółty okrąg}
   SetColor(YELLOW);
   Circle(300,200,100);

   {będziemy wypełniać na czerwono)
   SetFillStyle(2,RED);

   {wypełniamy obszar (okrąg) , podajemy dowolny
   punkt, który leży wewnątrz danej figury (np.
   środek okręgu) i kolor obramowania danej figury)
   FloodFill(300,200,YELLOW);

   ReadLn;
   CloseGraph;
end.
```

11.4. Napisy

Moduł Graph udostępnia instrukcję OutTextXY, która pozwala na wyświetlanie dowolnych napisów. Oto jej składnia:

```
procedure OutTextXY(X,Y : Integer; Napis : String);
```

Jak widzimy, oczekuje ona trzech argumentów. Pierwsze dwa, to współrzędne na ekranie, gdzie ma być wyświetlany napis, a jego treść przekazujemy w trzecim argumencie. Oto przykład:

```
program PO 8 5;
uses Crt,Graph;
var Karta,Tryb : Integer;
    i : Integer;

begin
   Randomize;
   DetectGraph(Karta,Tryb) ;
   InitGraph(Karta,Tryb,'c:\bp\bgi');

   for i:=1 to 2 0 do
   begin
     SetColor(Random(15)+1);
     SetTextStyle(Random(10),Random(2),Random(10));
     OutTextXY(Random(GetMaxX-100), M
               Random(GetMaxY-100),'Turbo Pascal');
   end;

   ReadLn;
   CloseGraph;
end.
```

Turbo Pascal posiada 10 różnych krojów czcionek. Wybór czcionki, jak również jej wielkość i sposób wyświetlania określamy procedurą SetTextStyle. Wykorzystaliśmy ją w powyższym programie. Oto jej składnia:

```
procedure  SetTextStyle(Czcionka,Kierunek,
                        Wielkość : Integer);
```

Jako rodzaj czcionki podajemy liczbę z zakresu od 1 do 10. ICierunek, to dwie wartości: 0 - dla tekstu poziomego i 1 - dla tekstu pionowego.

12. Zakończenie

No i nadszedł koniec naszej książki, co oczywiście nie oznacza, że wyczerpaliśmy wszystkie kwestie związane z Turbo Pascalem. Jeśli ten język spodobał Ci się choć trochę możesz oczywiście poświęcić mu jeszcze wiele uwagi, sięgnąć po kolejne lektury, zajrzeć do zasobów Internetu. Pomimo, iż Pascal jest językiem prostym w nauce, jego możliwości są bardzo duże. Nie sposób było omówić wszystkiego na łamach tej książki. Szczególnie, że adresowana jest ona dla „nieinformatyków". Dlatego moim zamiarem było przedstawienie Ci podstaw języka tak, abyś mógł samodzielnie stworzyć proste programy, czy też zechciał sięgnąć po dalszą literaturę. Chcąc Ci w tym pomóc w Dodatku A umieściłem wykaz tytułów i adresów WWW, gdzie można szukać takich informacji. Szczególnie polecam zajrzenie właśnie do stron internetowych traktujących na temat Pascala. Można tam odnaleźć setki przykładowych kodów źródłowych, a z nich uczymy się najszybciej. Czynimy to czytając treść programów i je analizując.

To jest pierwsze wydanie tej książki. Znajdzie się w niej z pewnością kilka pomyłek, czy też wpadek. Być może omówiłem jakiś temat w sposób nie najbardziej przyjazny czytelnikowi. Być może, w niektórych miejscach użyłem zbyt wiele słów technicznych, przez co treść stała się mało zrozumiała. Bywa tak często, ponieważ czasami zapominamy, że tłumaczymy coś osobom początkującym. Dlatego byłbym Ci bardzo wdzięczny za wyrażenie własnej opinii na temat tej lektury. Na wskazanie tych miejsc, z którymi miałeś najwięcej kłopotów. W ten sposób możesz pomóc w stworzeniu kolejnych wydań, znacznie lepiej przedstawiających zawiłości języka Pascal. Tak pomożesz również kolejnym nowicjuszom, którzy czy to z pasji, czy też z obowiązku sięgną po tę pozycję. Na wszystkie komentarze czekam pod adresem e-mail: *binboy@binboy.org*.

Dziękuję za spędzony ze mną czas.

Karol Wierzchołowski

Dodatek A - dowiedz się więcej

Internet to prawdziwy wehikuł wiedzy. Można w nim znaleźć bardzo wiele informacji na prawie każdy temat, oczywiście również na temat Turbo Pascala. Poniżej zamieściłem listę kilku adresów do stron WWW godnych polecenia. Znajdziesz na nich oprócz różnego rodzaju kursów, artykułów, również setki przykładowych kodów źródłowych, opisów algorytmów, itd.

- *http://www.binboy.org* - Portal Programisty
- *http://www.borland.pl* - Borland

Chciałbym również polecić kilka książek na temat Pascala.

- **„Prosto do celu PASCAL Ćwiczenia",** Karol Wierzchołowski, wyd. *ReadMe*
- **„Pascal. Wprowadzenie do programowania",** Wiesław Porębski, *Komputerowa Oficyna Wydawnicza „HELP"*
- **„DELPHI - SAMOUCZEK",** Karol Wierzchołowski, *Komputerowa Oficyna Wydawnicza „HELP"*

Dodatek B - usuń usterkę

Kiedy tworzony był kompilator i całe środowisko do Turbo Pascala w wersji 7.0 procesory były znacznie wolniejsze. Sięgały niewiele ponad 100 MHz. Stąd też projektanci nie przewidzieli, że zaledwie po kilku latach procesory taktowane zegarami kilkuset MHz będą mogły zaszkodzić aplikacjom napisanym w Pascalu.

Usterka znalazła się w module Crt. Jeśli nasz program korzysta z niego, na szybkich komputerach nie wykona się prawidłowo. Wyświetli się natomiast komunikat o błędzie dzielenia przez zero. To trochę mylący komunikat, gdyż faktyczny błąd polega trochę na czymś innym, co wiąże się z tym komunikatem pośrednio. Nie wnikając w techniczne zawiłości problemu, musimy po prostu zainstalować sobie specjalnego patcha, eliminującego usterkę.

Owego patcha można pobrać z Internetu, ze strony:

- *http://binboy.org/index.php ?show=p200. htm*

Po ściągnięciu archiwum bp7.zip odpakuj jego zawartość do katalogu c:\bp\bin. Następnie przejdź do tego katalogu i uruchom program patch.exe. Wyświetlony zostanie monit. Wpisz: patch700 i wciśnij [ENTER]. Poprawki zostaną naniesione. Możesz już korzystać z modułu Crt nawet na procesorach osiągających prędkości kilku GHz.

Dodatek C - kody ASCII

DEC	HEX	znak	ctri	DEC	HEX	znak	DEC	HEX	znak	DEC	HEX	znak
0	00	NUL	A @	32	20	SP	64	40	@	96	60	v
1	01	SQH	A A	33	21	i	65	41	A	97	61	a
2	02	STX	A B	34	22	"	66	42	B	98	62	b
3	03	ETX	A C	35	23	#	67	43	C	99	63	c
4	04	EOT	A D	36	24	$	68	44	D	100	64	d
5	05	ENQ	A E	37	25	%	69	45	E	101	65	e
6	06	ACK	A F	38	26	&	70	46	F	102	66	f
7	07	BEL	A G	39	27	1	71	47	G	103	67	g
8	08	BS	A H	40	28	(	72	48	H	104	68	h
9	09	HT	A I	41	29	)	73	49	I	105	69	i
10	0A	LF	A J	42	2A	*	74	4A	J	106	6A	j
11	OB	VT	A K	43	2B	+	75	4B	K	107	6B	k
12	OC	FF	A L	44	2C	>	76	4C	L	108	6C	1
13	OD	CR	A M	45	2D	-	77	4D	M	109	6D	m
14	OE	SO	A N	46	2E		78	4E	N	110	6E	n
15	OF	SI	A 0	47	2F	/	79	4F	0	111	6F	0
16	10	DLE	A P	48	30	0	80	50	P	112	70	P
17	11	DCI	A Q	49	31	1	81	51	Q	113	71	q
18	12	DC2	A R	50	32	2	82	52	R	114	72	r
19	13	DC3	A S	51	33	3	83	53	S	115	73	s
20	14	DC4	AJ[1]	52	34	4	84	54	T	116	74	t
21	15	NAK	A U	53	35	5	85	55	U	117	75	u
22	16	SYN	A V	54	36	6	86	56	V	118	76	v
23	17	ETB	A W	55	37	7	87	57	W	119	77	w
24	18	CAN	A X	56	38	8	88	58	X	120	78	x
25	19	EM	A Y	57	39	9	89	59	Y	121	79	y
26	1A	SUB	A Z	58	3A		90	5A	Z	122	7A	z
27	IB	ESC	A [	59	3B	»	91	5B	[	123	7B	{
28	1C	FS	A \	60	3C	<	92	5C	\	124	7C	1
29	ID	GS	A]	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	AA	62	3E	>	94	5E	A	126	7E	~
31	IF	US	A	63	3F	?	95	5F		127	7F	DEL