

*Katedra Podstaw Informatyki*

# **Laboratorium z Algorytmów i Programowania**

Zadanie 6: Pascal

Karol Wierzchołowski

Gdańsk 2002

## Zadanie 6

### Pascal

#### Informacje ogólne

Pascal jest językiem wysokiego poziomu. Został zdefiniowany i opisany przez *Niklausa Wirtha* na uniwersytecie w Zurychu w 1968 roku. Wzorowany był na języku *Algol 60* stąd tak jak i on – zorientowany jest na programowanie strukturalne. Pierwszy kompilator Pascala powstał już dwa lata po jego opracowaniu – w roku 1970.

Pascal jest językiem idealnym jeśli chodzi o edukację. Niezwykle przejrzysty i czytelny kod sprawia, że debugowanie i jego analiza nie jest zadaniem tak żmudnym i trudnym jak w przypadku wielu innych języków. Przyswojenie nowych słów kluczowych, składni instrukcji, czy bogatych bibliotek procedur i funkcji nie sprawia początkującym większych problemów. Pascal jest językiem intuicyjnym, pozwalającym na stworzenie niemalże każdego programu. Można w nim tworzyć bazy danych, programy wykonujące skomplikowane obliczenia matematyczne, ciekawe efekty graficzne, czy nawet gry. Pascal (a w zasadzie kompilatory Pascala) posiada jednak również i wady. Główną z nich jest szybkość. Z reguły programy napisane w C działają kilkukrotnie szybciej od ich wersji w Pascalu – co w niektórych przypadkach może go zupełnie dyskwalifikować. Problemy mogą również pojawić się z przenośnością kodu na inne platformy sprzętowe, czy systemowe.

Istnieje wiele firm projektujących coraz to nowsze i szybsze kompilatory do tego języka. Najbardziej znaną jest firma Borland, która wypuściła już pakiety *Turbo Pascala* do wersji 7.0. Odbiegają one znacznie od pierwotnego Pascala. W 1990 roku, kiedy powstał *Turbo Pascal 5.5*, wprowadzony został do niego typ obiektowy. Dzięki temu programowanie strukturalne mogło być stosowane łącznie i zamiennie z programowaniem obiektowym. Dalsze prace firmy nad językiem doprowadziły do powstania rozbudowanego środowiska programistycznego pod Windows – Delphi (rok 1995). Delphi jest językiem opartym na *Object Pascal*, pozwalającym na pisanie bardzo rozbudowanych aplikacji pod systemem Windows. Powstało już 7 wersji tego pakietu. W chwili obecnej dostępny jest również Kylix – czyli środowisko programistyczne bazujące na Delphi (a więc i *Object Pascal*) pod system Linux.

Oprócz *Turbo Pascala* firmy Borland dostępne są jeszcze implementacje innych firm. Najbardziej znane i godne polecenia to *TMT Pascal*, *FPC*, *Irie Pascal*. Są to kompilatory darmowe i dostępne pod wieloma różnymi platformami sprzętowymi i systemami operacyjnymi.

Godnymi polecenia pozycjami książkowymi na temat Turbo Pascala są:

1. *Wstęp do programowania z przykładami w Turbo Pascalu*, Koleśnik K., Helion 1999
2. *Turbo Pascal 7.0*, Marciniak A., Nakom

Przydatne adresy WWW:

1. <http://www.borland.pl/> - Borland Pascal
2. <http://www.freepascal.org/> - Free Pascal
3. <http://www.tmt.com/> - TMT Pascal
4. <http://www.iriertools.com/> - Irie Pascal

W dalszej części tej pracy będzie używane zamiennie określenie Pascal i Turbo Pascal (wersja rozszerzona firmy Borland). Wszystkie przykłady powinny jednak kompilować się bez żadnych błędów także pod innymi kompilatorami poleconymi wcześniej.

## Kompilacja

Kompilacja programów wygląda trochę inaczej w każdym z poleconych wcześniej kompilatorów. Omówione zostaną dwa z nich. Pakiet *Turbo Pascal* oraz *TMT Pascal*.

### Turbo Pascal

W przypadku pakietu *Turbo Pascal* – edycja, kompilacja i testowanie – wykonywane jest z jednego miejsca. Turbo Pascal to zintegrowany edytor, kompilator i debugger. Po jego uruchomieniu (polecenie `bp.exe` lub `turbo.exe`) na ekranie wyświetlony zostanie niebieski ekran edytora. W nim wpisujemy treść programu. Menu jest niezwykle intuicyjne i jego poznanie nie powinno sprawić kłopotów.

Aby skompilować program znajdujący się w aktywnym okienku edytora wciśnij klawisz F9 lub z menu `COMPILE` wybierz opcję `COMPILE`. Zostanie wyświetlone wówczas okno z postępowaniem kompilacji. Jeśli wystąpią jakieś błędy wyświetlony zostanie odpowiedni komunikat, a kursor ustawiony w odpowiednie miejsce. W przypadku gdy kompilacja zakończy się sukcesem, można uruchomić dany program korzystając z kombinacji klawiszy `CTRL+F9` lub z menu `RUN` wybrać pozycję `RUN`.

Inne przydatne skróty klawiszowe:

- `ALT+F5` – podgląd okna wykonywania programu
- `ALT+numer` – przełączanie się między oknami
- `ALT+F3` – zamknięcie okna
- `F2` – zapisanie dokumentu
- `F3` – odczytanie dokumentu
- `F8` i `F7` – praca krokowa (uruchamianie programu linia po linii)

### TMT Pascal

Środowisko TMT Pascal możemy ściągnąć bezpośrednio ze strony domowej <http://www.tmt.com/> (gdzie dostępne jest także środowisko programistyczne wraz z edytorem i debuggerem), lub ze strony BHP (<http://www.binboy.org> dział `Download/Kompilatory/Pascal`). W serwisie BHP dostępna jest wersja 3.0 tego pakietu, a więc nie najnowsza. Nadaje się jednak w zupełności do naszych potrzeb i nie ma konieczności wypełniania żadnych formularzy.

Po pobraniu wersji archiwalnej należy rozpakować ją do dowolnego katalogu (np. `tmt`). Obsługa kompilatora odbywa się z wiersza poleceń. Wszystkie polecenia (kompilator, konsolidator, etc.) znajdują się w katalogu `bin`, a więc należy dodać odpowiednią ścieżkę do zmiennej `PATH` lub przebywać w tym katalogu.

Programy można pisać w dowolnym edytorze ASCII, np. Notepad, czy Edit. Pliki zapisujemy z rozszerzeniem `*.pas`.

Aby skompilować program wydajemy polecenie:

```
tmtpc nazwa_pliku.pas
```

W wyniku działania kompilatora i przy braku błędów powstanie na dysku plik `nazwa_pliku.exe`, który możemy uruchomić. Wydanie polecenia `tmtpc` bez żadnych parametrów spowoduje wyświetlenie listy opcji, których możemy użyć przy kompilacji.

Kompilator TMT Pascal jest bardzo rozbudowanym narzędziem. Potrafi optymalizować kod pod procesory obsługujące MMX, czy nawet 3DNow!. Jest to produkt darmowy przy wykorzystaniu niekomercyjnym. Kompilowane programy pracują domyślnie w trybie chronionym

## Wstęp do Pascala

Celem laboratorium jest zapoznanie Czytelnika z językiem Pascal oraz zilustrowanie na jego przykładzie wspólnych cech wszystkich języków strukturalnych. Zakładając u Czytelnika znajomość podstaw języka C, odwołamy się do tych wiadomości i na zasadzie analogii przedstawione zostaną podstawowe elementy Pascala.

## Struktura programu

Oto prosty program pokazujący jak wygląda szablon programu w Pascalu:

```
PROGRAM Przyklad;  
USES Crt;  
BEGIN  
  ClrScr;  
  Writeln('AIP - Algorytmy i Programowanie');  
END.
```

### Uwaga!

W Pascalu wielkość liter nie ma znaczenia. Możesz więc pisać zarówno BEGIN, jak i begin, czy BeGiN.

Wyłuszczone słowa – to tzw. *słowa kluczowe*. Mają one swoje predefiniowane znaczenie i należy ich używać zawsze w określony sposób (podobnie jak słowa kluczowe w C).

Każdy poprawnie napisany program rozpoczyna się słowem kluczowym PROGRAM, po którym umieszczamy nazwę programu. Nazwa ta – podobnie jak nazwa zmiennej, funkcji, procedury, czy nowego typu – jest tzw. *identyfikatorem*. Może więc się składać tylko z liter, cyfr i znaku podkreślenia (przy czym nie może zacząć się cyfrą, ani zawierać polskich znaków diakrytycznych). Linia ta nie jest obowiązkowa, lecz dobrym stylem jest jej stosowanie.

Kolejny blok programu to tzw. *deklaracja modułów*. Moduły są to skompilowane (lub w wersji ze źródłami) biblioteki gotowych funkcji i procedur. Są więc czymś podobnym do plików nagłówkowych w C (choć nie do końca). Deklaracja modułów także nie jest obowiązkowa. Wiele funkcji i procedur znajduje się w tzw. module SYSTEM, który dołączany jest do programu automatycznie. Zawiera on implementację podstawowych instrukcji do wyświetlania napisów, wykonywania operacji matematycznych, itp. Inne – często wykorzystywane moduły to:

1. CRT  
Zawiera szereg procedur i funkcji do grafiki tekstowej, a więc np. procedurę ClrScr – służącą do czyszczenia ekranu, GotoXY – do ustawiania kursora, TextColor, TextBackGround – do zmiany koloru liter i tła, itp. Jest więc odpowiednikiem niestandardowego pliku nagłówkowego CONIO.H.
2. DOS  
Moduł zawiera szereg procedur i funkcji specyficznych dla danego systemu operacyjnego, np. wyszukiwanie pliku, tworzenie katalogu, itd.
3. GRAPH  
Moduł zawiera zestaw gotowych funkcji do obsługi trybu graficznego, a więc procedury do rysowania okręgów, linii, wypełniania obszarów, wyświetlania napisów, itp.

Po deklaracji modułów znajduje się tzw. *program główny*. Jest to ciąg instrukcji mieszczący się pomiędzy słowami BEGIN i END. (END zakończone kropką). Słowa BEGIN i END są odpowiednikami nawiasów klamrowych w języku C.

Działanie programu rozpoczyna się zawsze od tego miejsca (podobnie jak w przypadku C – od funkcji main).

W naszym przykładzie program składa się z dwóch instrukcji. Pierwszą jest wywołanie procedury ClrScr – która służy do wyczyszczenia ekranu, drugą – wywołanie procedury Writeln – służąca do wyświetlenia napisu na ekranie. Zwróć uwagę, że po instrukcji ClrScr nie ma nawiasów, jest natomiast średnik. Brak nawiasów podyktowany jest tym, że procedura ta nie potrzebuje żadnych argumentów. W języku C – nawet kiedy funkcja nie pobierała argumentów – należało pisać puste nawiasy.

### Uwaga!

W języku Pascal każda instrukcja kończy się średnikiem.

## Stałe i zmienne

W Pascalu do deklaracji stałych służy słowo kluczowe `CONST`. Deklaracja wygląda w następujący sposób:

```
CONST ident = 12;  
      X     = 3.14;  
      nap   = 'AIP';
```

Po słowie `CONST` znajduje się identyfikator stałej, a po znaku równości jej wartość. Może to być liczba całkowita, rzeczywista, czy napis. Jeśli deklarujemy kilka stałych, deklaracje umieszczamy jedna pod drugą (jak w przykładzie). Nie ma obowiązku poprzedzania każdej z nich słowem `CONST`.

### Uwaga!

W Pascalu wszystkie napisy umieszczamy w apostrofach  
(nie w cudzysłowach, jak ma to miejsce w C)

Deklarację zmiennych rozpoczynamy słowem kluczowym `VAR`. Oto przykład:

```
VAR x : Integer;  
    y : Byte;  
    Napis : STRING;  
    Krotki : STRING[10];  
    Ulamek : Real;  
    a,b,c : Word;
```

Jak widać – po słowie `VAR` piszemy identyfikator zmiennej, a następnie po dwukropku jej typ. Jeśli deklarujemy kilka zmiennych tego samego typu to możemy wymienić je po przecinku. Kolejne deklaracje umieszczamy w kolejnych liniach. Podobnie jak w przypadku stałych - kolejne linie deklaracji nie poprzedzamy już słowem `VAR`.

## Podstawowe operatory i typy danych

Każda zmienna jest określonego typu (podobnie jak w C). Pascal daje nam do dyspozycji bardzo dużo wbudowanych typów danych. Podstawowe z nich to (w nawiasie podano odpowiednik w języku C):

- `BYTE` – krótka liczba całkowita dodatnia (**unsigned char**)
- `SHORTINT` – krótka liczba całkowita (**signed char**)
- `INTEGER` – liczba całkowita (**int**)
- `WORD` – liczba całkowita dodatnia (**unsigned int**)
- `LONGINT` – długa liczba całkowita (**long int**)
- `CHAR` – typ znakowy (**char**)
- `REAL` – liczba rzeczywista (**float**)
- `BOOLEAN` – typ logiczny (nie ma odpowiednika)
- `STRING` – typ napisowy (nie ma odpowiednika)

Z deklaracją zmiennej danego typu wiążą się dwie rzeczy: ilość zajmowanej przez nią pamięci oraz rodzaj możliwych do wykonania na niej operacji.

Na zmiennych całkowitych możemy wykonywać podstawowe operacje matematyczne – a więc dodawanie (+), odejmowanie (-), mnożenie (\*). Można także wykonać tzw. dzielenie całkowite (`DIV`) oraz dzielenie modulo (reszta z dzielenia), czyli operator `MOD`. Na zmiennych rzeczywistych można wykonywać wszystkie operacje arytmetyczne – a więc dodawanie, odejmowanie, mnożenie i dzielenie (/). Nie można natomiast wykonać dzielenia całkowitego, ani modulo. W C było trochę inaczej. Był tam operator `“/”`, który działał jako dzielenie całkowite – gdy były zmienne całkowite – i jako dzielenie rzeczywiste gdy były liczby rzeczywiste. Operator dzielenia modulo w języku C pisany był jako `“%”`.

Zmienne typu logicznego mogą przyjmować tylko dwie wartości: TRUE (prawda) oraz FALSE (fałsz). Nie mają swojego odpowiednika w C. Tam prawdę oznaczała dowolna wartość różna od zera, natomiast fałsz – zero. W Pascalu jak widzimy jest nieco inaczej.

Pascal daje użytkownikowi nowy typ danych, mianowicie typ łańcuchowy STRING. Służy on do przetrzymywania i manipulowania ciągami znaków (napisami). Zmienne tego typu widziane są przez kompilator jako tablice znaków, do których mamy dostęp przez tzw. *indeks* (a więc podobnie jak w C). W bajcie pod zerowym indeksem zapisana jest długość ciągu (przypomnimy, że w języku C długość nie była nigdzie zapisana, a o końcu tekstu świadczył kod `'\0'`), natomiast w kolejnych bajtach poszczególne litery.

Przy deklaracji zmiennej łańcuchowej w nawiasie kwadratowym możemy określić maksymalną długość ciągu (ilość bajtów potrzebnych do zapamiętania). Nie może to być liczba większa niż 255.

Ponieważ łańcuchy to “normalne” typy danych, a nie tablice (formalnie) – możemy na nich wykonywać podstawowe operacje, np. przypisywanie, kopiowanie, czy sklejanie (operator +). Takie czynności w języku C były niemożliwe.

Poniżej znajduje się przykładowy program demonstrujący użycie zmiennych różnego typu w programie:

```
PROGRAM TypyDanych;
USES Crt;
CONST stala = 'Binboy';
VAR a,b : STRING;
    x,y : Integer;
    q,w : Real;
    S   : BOOLEAN;
BEGIN
  ClrScr;
  X:=4;
  Y:=X+12-(X DIV 2) MOD 2;
  Q:=3.14;
  W:=Q/12*7;
  S:=TRUE;
  A:='AIP';
  B:=A+' [Algorytmy i programowanie]';
  Writeln('Stała: ',stala);
  Writeln('Napis: ',B);
  Writeln('Liczba całkowita: ',Y);
  Writeln('Liczba rzeczywista: ',W);
  Writeln('Napis A=',A,' jego długość=',Ord(A[0]));
END.
```

#### Uwaga!

Operator przypisania w Pascalu to sekwencja `:=`, a nie sam znak równości. Pojedynczy znak `=` wykorzystywany jest przy porównywaniu (odpowiednik `==` z C)

Zwróćmy uwagę na składnię instrukcji `Writeln`. Pascal nie posiada odpowiednika wiernego funkcji `printf`. Do wyświetlania napisów służą dwie procedury: `Writeln` i `Write`. Użycie ich jest identyczne. Różnią się jedynie tym, że pierwsza z nich automatycznie przenosi kursor do nowej linii (jak `printf("\n");`).

Zauważmy również w jaki sposób odwołał się do zerowego elementu napisu A (jak do tablicy w C). Ponieważ zerowy element tablicy określa długość napisu, ale jako element napisu – jest literą nie mogliśmy go wyświetlić “normalnie”. Dlatego za pomocą funkcji `Ord` zamieniliśmy kod ASCII tej litery (a więc faktyczną długość ciągu) na liczbę i wyświetliliśmy ją na ekranie.

W Pascalu wszystkie operacje arytmetyczne i logiczne wykonywane są od lewej strony do prawej zgodnie z priorytetami poszczególnych operatorów, które zebrano w tabeli poniżej:

Priorytet	Operatory
1	NOT
2	* / DIV MOD AND
3	+ - OR
4	< <= = <> >= >

Wyrażenia zapisane w nawiasach wykonywane są w pierwszej kolejności. Nawiasy można dowolnie zagnieżdżać.

## Instrukcje warunkowe

### Instrukcja IF

Instrukcje warunkowe w Pascalu są bardzo podobne do ich odpowiedników w C. Oto ogólna składnia warunku IF:

```
IF warunek THEN instrukcja [ELSE instrukcja];
```

Po słowie kluczowym IF umieszczamy warunek naszej instrukcji. Warunek musi przyjmować jedną z dwóch wartości logicznych: TRUE albo FALSE. Następnie znajduje się zawsze słowo kluczowe THEN, po którym umieszczamy jedną instrukcję (lub blok instrukcji) zakończoną średnikiem. Ponieważ warunek znajduje się pomiędzy dwoma słowami kluczowymi, nie musimy umieszczać go w nawiasach (jak ma to miejsce w przypadku języka C). Proszę zwrócić uwagę na blok zapisany w nawiasie kwadratowym (tak oznaczamy blok opcjonalny). Użycie instrukcji warunkowej w rozszerzonej formie (z ELSE – czyli “w przeciwnym wypadku”) polega na usunięciu tych nawiasów z szablonu składni. A więc – po słowie ELSE umieszczamy kolejną instrukcję i kończymy całość średnikiem. Zwróćmy również uwagę, że przy wersji rozszerzonej nie ma średnika po instrukcji znajdującej się za słowem THEN.

### Uwaga!

Umieszczanie średnika po instrukcji za słowem THEN w wersji rozszerzonej warunku IF jest bardzo częstym błędem popełnianym przez osoby wcześniej programujące w języku C.

W bloku warunkowym instrukcji IF możemy umieścić kilka warunków oddzielając je odpowiednim operatorem logicznym. Każdy z takich podwarunków umieszczamy w nawiasach. Oto dostępne operatory (w nawiasie ich odpowiedniki z C):

- **OR** – “lub” – alternatywa (“|”)
- **AND** – “i” – koniunkcja (“&”)
- **NOT** – “nie” – negacja (“!”)

Oto przykład:

```
IF (x>=10) AND (x<=20) THEN Writeln('X z przedziału 10..20')
    ELSE Writeln('X spoza przedziału');
```

Przy porównywaniu możemy korzystać z następujących operatorów:

- = - czy równe
- > - czy większe
- >= - czy większe lub równe
- < - czy mniejsze
- <= - czy mniejsze lub równe
- <> - czy różne

Zwróć uwagę, że operatory są podobne do ich wersji z języka C. Różnią się jedynie: operator “różności” (w C jest to “!=”), oraz operator równości (w C “==”)

#### **Uwaga!**

Osoby programujące w C mogą często popełniać błąd w wykorzystaniu operatora równości. W Pascalu jest to tylko jeden znak “równa się”, co w C miało zupełnie inne znaczenie. Proszę zwrócić na to uwagę!

#### **ZADANIE (1 pkt)**

Napisz program wczytujący trzy współczynniki całkowite i drukujący pierwiastki całkowite równania  $ax^2+bx+c=0$  z dokładnością do 5 miejsc po przecinku.

#### **Instrukcja CASE**

Odpowiednikiem instrukcji SWITCH z języka C jest CASE w Pascalu. Składnia jest bardzo podobna:

```
CASE wyrażenie OF  
wartosc_1: instrukcja;  
wartosc_2: instrukcja;  
...  
[ELSE: instrukcja]  
END;
```

Na początku umieszczamy słowo kluczowe CASE (w C pisaliśmy SWITCH), po którym znajduje się wyrażenie przyjmujące wartości całkowite. Po nim piszemy słowo OF. I tak jak w przypadku warunku IF – ponieważ wyrażenie znajduje się pomiędzy dwoma słowami kluczowymi nie musimy umieszczać go dodatkowo w nawiasach. W kolejnych liniach umieszczamy wartości, jakie może przyjąć dane wyrażenie, a po dwukropku instrukcje do wykonania. Opcjonalnie możemy umieścić jeszcze linię z ELSE (odpowiednik default z C), która zostanie wykonana, jeśli wyrażenie ma inną wartość niż te wszystkie podane na liście. Całość zawsze kończymy słowem END.

W instrukcji CASE, kiedy wyrażenie przyjmie daną wartość wykonana zostanie tylko dana instrukcja. W C było inaczej. Wykonywane były również wszystkie instrukcje znajdujące się poniżej. Proszę zwrócić na to uwagę!

#### **Instrukcje iteracyjne**

Działanie instrukcji iteracyjnych jest niemalże identyczne. Różnica polega tylko na formalnym zapisie.

#### **Pętla FOR**

Składnia jest następująca:

```
FOR licznik:=wart_pocz TO wart_konc DO instrukcja;  
lub dla pętli malejącej:  
FOR licznik:=wart_konc DOWNTO wart_pocz DO instrukcja;
```

#### **Uwaga!**

Licznik musi być zmienną typu całkowitego.

Pętla FOR w Pascalu jest więc uboższą wersją tej z języka C. Narzuconych jest też wiele wymagań, których nie sposób ominąć. Nie można np. stworzyć pętli “skaczącej” o 2, lub pominąć któregoś z parametrów. Niemniej jednak jest to instrukcja niezwykle często wykorzystywana i przydatna.

Oto przykładowy program wyświetlająca na ekranie 10 liczb:

```
PROGRAM Petle;
VAR i : Integer;
BEGIN
  FOR i:=1 TO 10 DO Writeln(i);
END.
```

### ZADANIE (2 pkt)

Napisz program wyświetlający trójkąt prostokątny o wymiarach a x b zgodnie z rysunkiem:

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

### Pętla WHILE

Jej użycie i składania jest niemalże identyczna jak jej odpowiednik z C:

```
WHILE warunek DO instrukcja;
```

Instrukcja jest wykonywana dopóty, dopóki warunek jest spełniony (przyjmuje wartość TRUE).

### Pętla REPEAT ... UNTIL

Jej odpowiednikiem w C jest instrukcja DO . . WHILE. Oto składnia:

```
REPEAT
  Instrukcja;
  ...
  Instrukcja;
UNTIL warunek;
```

Pętla jest wykonywana dopóki warunek nie jest prawdziwy. Bardzo często jest wykorzystywana w połączeniu z funkcją `KeyPressed` – kiedy chcemy wykonywać dany blok programu dopóki użytkownik nie wciśnie żadnego klawisza na klawiaturze, np.:

```
REPEAT
  Writeln('http://binboy.org');
UNTIL KeyPressed;
```

### ZADANIE (2 pkt)

Napisz program - Latające literki. Ekran powinien być podzielony na cztery ćwiartki (podziału ma nie być widać). W pierwszej ćwiartce ma “latać” gwiazdka – odbijając się cały czas od krawędzi. Za sobą ma zostawiać ślad o długości 10 gwiazdek. Symetrycznie do niej – w pozostałych ćwiartkach mają poruszać się literki “A”, “I” i “P”.

### Instrukcja złożona

W Pascalu (podobnie jak w C) istnieje pojęcie tzw. *instrukcji złożonej*. Mianem tym określamy dowolny blok instrukcji umieszczony między słowami kluczowymi BEGIN i END (przypomnimy, że w C blok był objęty nawiasami klamrowymi). Instrukcja złożona z punktu widzenia kompilatora widziana jest jako pojedyncza instrukcja. Możemy więc stosować ją wszędzie tam, gdzie składnia użycia danego elementu wymaga pojedynczej instrukcji. Oto przykład wykorzystania instrukcji w pętli FOR:

```

PROGRAM Petle2;
USES Crt;
VAR i : Integer;
BEGIN
  FOR i:=1 DO 10 DO
    BEGIN
      GotoXY(i+5, I+5);
      Writeln(i);
    END;
  END.

```

A oto przykład użycia instrukcji złożonej w instrukcji warunkowej:

```

...
IF Delta>0 THEN
BEGIN
  X1:=(-B-SQRT(Delta))/(2*A);
  X2:=(-B+SQRT(Delta))/(2*A);
END
ELSE IF (Delta=0) THEN X:=(-B)/(2*A)
ELSE Writeln('NIE MA PIERWIASTKÓW');

```

Proszę zwrócić uwagę, że po słowie END nie ma średnika. Podyktowane jest to tym, że w składni instrukcji warunkowej jest wyraźnie zaznaczone, że średnik jest jeden i znajduje się na końcu całej instrukcji. Ponieważ blok instrukcji złożonej widziany jest przez kompilator jako pojedyncza instrukcja obowiązują tutaj te same reguły. W przykładzie wcześniejszym (z pętlą FOR) średnik po END był konieczny.

## Procedury i funkcje

Funkcja (procedura) jest to pewien podprogram wykonujący określony ciąg instrukcji, zamknięty w pewnym bloku i opatrzony nazwą (identyfikatorem), przy pomocy którego możemy ją wywołać. Może ona przyjmować pewne argumenty, które następnie wykorzysta przy wykonywaniu kodu lub nie. W C nie istniało pojęcie procedury. Występowały tam jedynie funkcje. Procedura różni się od funkcji tym, że nie może zwrócić żadnej wartości. Jest to więc odpowiednik funkcji zwracającej wartość `void` w języku C. Deklaracja procedury i funkcji w Pascalu ma jednak inną składnię.

### Deklaracja funkcji

```

FUNCTION nazwa(argumenty) : typ_zwracanej_wartosci;
BEGIN
  ...
  nazwa:=wartosc;
END;

```

Jak widać – po słowie kluczowym FUNCTION umieszczamy nazwę (identyfikator) nowej funkcji. Następnie w nawiasach podajemy listę argumentów, których funkcja wymaga przy wywołaniu. Za nawiasem po dwukropku podajemy typ zwracanej przez funkcję wartość. Całość kończymy średnikiem. Dalej znajduje się blok instrukcji (BEGIN...END), czyli ciało danej funkcji, w którym umieszczamy dowolne instrukcje. Wartość zwracanej przez funkcję wartości określamy poprzez przypisanie jej do identyfikatora nazwy (zob. przykład).

```

FUNCTION kwadrat(X : Integer) : Integer;
BEGIN
  Kwadrat:=X*X;
END;

```

Powyższa funkcja zwraca kwadrat liczby podanej jako argument. Zwróć uwagę na sposób określania listy argumentów. Wygląda on podobnie jak deklaracja zmiennych, a więc najpierw identyfikator, a następnie po

dwukropku typ. Jeśli chcielibyśmy na liście podać kilka argumentów, należałoby oddzielić je średnikami. Jeśli mają być one tego samego typu – możemy wymienić je po przecinku. Oto przykład:

```
FUNCTION wykonaj(X,Y : Integer; W : STRING) : Real;
...
```

#### Uwaga!

W języku C wszystkie argumenty oddzielaliśmy przecinkami – w Pascalu oddzielamy je średnikami, a przecinkami oddzielamy argumenty tego samego typu.

### Deklaracja procedury

Oto składnia:

```
PROCEDURE nazwa(argumenty);
BEGIN
...
END;
```

Deklaracja procedury jest uboższą wersją deklaracji funkcji. Różnice polegają na tym, że używamy słowa kluczowego `PROCEDURE` i nie podajemy typu zwracanej wartości.

#### Uwaga!

Jeżeli nie przekazujemy argumentów do funkcji/procedury .pomijamy nawiasy występujące po nazwie (nie zostawiamy ich pustych jak to ma miejsce w przypadku języka C)

### Zmienne lokalne i przekazywanie parametrów do funkcji/procedury

Wewnątrz procedury lub funkcji możemy zadeklarować tzw. *zmienne lokalne*, a więc widoczne tylko w jej obrębie. Robimy to w ten sam sposób, jak przy deklaracji zmiennych globalnych – umieszczając blok deklaracyjny przed słowem `BEGIN`. Oto przykład:

```
PROGRAM ZmienneLokalne;

PROCEDURE napisz(tekst : STRING; Ilosc : Integer);
VAR i : Integer;
BEGIN
    FOR i:=1 TO Ilosc DO Writeln(tekst);
END;

BEGIN
    Napisz('[AIP] Algorytmy i programowanie',10);
END.
```

Lista argumentów zadeklarowana w sposób opisany wyżej – jest jednocześnie deklaracją zmiennych lokalnych, którym nadana zostanie wartość przekazana do funkcji/procedury. Jest to więc przekazywanie zmiennych przez wartość. Istnieje jednak jeszcze inna metoda przekazywania argumentów do podprogramów – mianowicie – przez adres. W tym celu daną deklarację w bloku argumentów poprzedzamy słowem kluczowym `VAR`. Oto przykład:

```
PROCEDURE Zmylka(VAR x : Integer);
BEGIN
    X:=5;
END;
...
Zmylka(Y);
```

W tym przykładzie nie tworzona jest zmienna lokalna `x`. Odwoływanie się do niej powoduje odwołanie się tak naprawdę do zmiennej `Y` i zmianę jej wartości.

## Tablice i rekordy

### Tablice

Do deklaracji tablicy w Pascalu zostało stworzone nowe słowo kluczowe `ARRAY`. Możemy zadeklarować zmienną jako tablicę bezpośrednio, lub stworzyć wcześniej nowy typ danych z wykorzystaniem tablicy i skorzystać z niego. Oto ogólna składnia deklaracji:

#### TYPE

```
TTablica = ARRAY[start..koniec] OF typ_podstawowy;
```

W ten sposób tworzymy nowy typ danych `TTablica`, który później możemy wykorzystać przy deklaracji zmiennej tablicowej, w następujący sposób:

```
VAR Tab : TTablica;
```

Po słowie kluczowym `ARRAY`, w nawiasach kwadratowych określamy rozmiar tablicy. Poszczególne wymiary oddzielamy przecinkami. Poniżej znajduje się przykład deklaracji zmiennej tablicowej mogącej przechować 6 napisów, indeksowanej od 5 do 10:

```
VAR Tab : ARRAY[5..10] OF STRING;
```

Do tablicy odwołujemy się poprzez indeks podawany w nawiasach kwadratowych (a więc identycznie jak w przypadku C). Zwróćmy tylko uwagę, że w Pascalu określamy przedziały indeksowania, a więc może to być (jak w naszym przypadku) od 5. W C tablice indeksowało się zawsze od zera. Jeśli zadeklarujemy tablicę wielowymiarową, aby określić pozycję danego elementu, przy odwoływaniu się do niego podajemy wszystkie wymiary oddzielając je przecinkami. Spójrzmy na przykład:

```
PROGRAM Tablice;
VAR Tabliczka : ARRAY[1..10,1..10] OF Integer;
    X,Y : Integer;
BEGIN
    FOR X:=1 TO 10 DO
        FOR Y:=1 TO 10 DO Tabliczka[X,Y]:=X*Y;

    Writeln('8 x 7 = ',Tabliczka[8,7]);

END.
```

Elementy tablicy mogą być dowolnego typu. Mogą to więc być również tablice. Inny przykład:

#### TYPE

```
Wiersz = ARRAY[1..10] OF Integer;
VAR Tabliczka : ARRAY[1..10] OF Wiersz;
```

Powyższa deklaracja jest równoważna tej z programu wcześniejszego. Aby odwołać się do pól tak zadeklarowanej tablicy należałoby podać współrzędne w sposób następujący:

```
Tabliczka[X][Y]:=X*Y;
```

W praktyce do obu tych tablic możemy odwoływać się przez podawanie indeksów po przecinku (zapis pierwszy) lub w oddzielnych nawiasach kwadratowych (zapis drugi). Jednak druga forma deklaracji tablicy pozwala nam się odwoływać do całego wiersza, natomiast pierwsza nie.

### Rekordy

Rekordy w Pascalu są tym, czym są struktury w języku C. Służą więc do grupowania kilku zmiennych pod jedną nazwą (jako nowy typ danych). Do deklaracji rekordów służy słowo kluczowe `RECORD`. Składnia deklaracji jest następująca:

**TYPE**

```

TRekord = RECORD
    Zmienna : Typ_danych;
    Zmienna : Typ_danych;
    ...
END;

```

Do pól w zmiennych typu rekordowym odwołujemy się poprzez operator kropki (bez względu, czy są to zmienne dynamicznie utworzone, czy też statycznie). Oto przykład:

**PROGRAM** Rekordy;**TYPE** TDane = RECORD

```

    Imie : STRING[20];
    Nazwisko : STRING[40];
    Telefon : Longint;
END;

```

**VAR** Osoba : TDane;**BEGIN**

```

    Write('Podaj imie>>');
    ReadLn(Osoba.imie);
    Write('Podaj nazwisko>>');
    ReadLn(Osoba.nazwisko);
    Write('Podaj telefon>');
    ReadLn(Osoba.telefon);

    Writeln('-----');
    Writeln('    Imie: ',Osoba.imie);
    Writeln(' Nazwisko: ',Osoba.nazwisko);
    Writeln('  Telefon: ',Osoba.telefon);

```

**END.****ZADANIE (2 pkt)**

Napisz program pobierający od użytkownika listę 10 nazwisk i ocen, a następnie wyświetlający wprowadzone dane w kolejności alfabetycznej oraz średnią danej grupy.

## Rekurencyjne typy danych i zmienne dynamiczne

### Zmienne dynamiczne

Wszystkie tworzone przez nas dotychczas zmienne, czy tablice – były strukturami statycznymi. Tworzone były więc automatycznie podczas działania programu. Wiemy jednak, że w wielu zastosowaniach algorytmicznych nie jest to rozwiązanie optymalne. Utrudnia (jeśli nie uniemożliwia) stworzenie bardziej zaawansowanych struktur danych takich jak listy, kolejki, stosy, drzewa, czy grafy. Pascal oczywiście pozwala na tworzenie zmiennych dynamicznych. Służy do tego celu procedura `New` (jest to więc odpowiednik funkcji `malloc` z języka C). Do zwalniania pamięci używamy instrukcji `Dispose` (odpowiednik `free` z C). Oto prosty przykład:

**VAR** Liczba : ^Integer;**BEGIN**

```

    New(Liczba);
    Liczba^:=5;
    Writeln(Liczba^);
    Dispose(Liczba);

```

**END.**

W C do określania zmiennych wskaźnikowych oraz do odwoływania się do wartości zapisanych pod wskazywanymi przez nie adresami – używaliśmy operatora gwiazdki “\*”. W Pascalu służy do tego celu *daszek* – “^”. Deklaracja zmiennej wskaźnikowej jest bardzo łatwa, po prostu jej typ poprzedzamy tym symbolem. Aby utworzyć zmienną dynamiczną wykorzystujemy procedurę *New*, której jako argument przekazujemy zmienną wskaźnikową. Zostanie w niej zapisany adres do przydzielonego obszaru pamięci. Aby odwołać się do tak utworzonej zmiennej (wskazywanej przez wskaźnik) również używamy operatora ^ - jak w przykładzie. Oczywiście można tworzyć wskaźniki do dowolnego typu danych. Możemy więc tworzyć tablice wskaźników do liczb całkowitych, wskaźniki do rekordów, itd. Oto przykład tworzący dynamicznie zmienną typu TDane:

```
VAR Osoba : ^TOsoba;
BEGIN
  New (Osoba);
  Osoba^.imie:='Karol';
  ...
  Dispose (Osoba);
  ...
```

Odpowiednikiem operatora & z języka C, służącego do pobierania adresu zmiennej w pamięci komputera, służy operator @. Oto przykład:

```
VAR X : Integer;
    Y : ^Integer;
BEGIN
  X:=2;
  Y:=@X;
  Writeln(Y^);
END.
```

### ZADANIE (2 pkt)

Napisz program, który prosi o wprowadzenie przez użytkownika 10 nazwisk, które zapamiętuje w liście jednokierunkowej, a następnie wyświetla je w kolejności alfabetycznej. Wprowadzanie danych do listy powinno zachowywać porządek.

### Rekurencyjne typy danych

Aby stworzyć listę, kolejkę, czy inną bardziej złożoną strukturę danych musimy wykorzystać tzw. rekurencyjność typów danych. Oznacza to, że w danym rekordzie chcemy stworzyć zmienną (pole) wskaźnikową do takiego samego typu. Oto przykład, który to obrazuje:

```
TYPE
  TDane = ^TOsoba;
  TOsoba = RECORD
    Imie : STRING[20];
    Telefon : Longint;
    Nastepny : TDane;
END;
```

## Struktura programu

Po przedstawieniu wszystkich elementów języka (typy danych, deklaracje stałych, zmiennych, itp.) można przedstawić pełny szablon struktury programu napisanego w Pascalu. Określa on dokładnie w jakiej kolejności powinny występować deklaracje w naszych programach. Czasami jednak można odbiegać od tej zasady. Oto on:

[PROGRAM Nazwa_programu;]	<b>PROGRAM</b> Test;
[Deklaracja modułów]	<b>USES</b> Crt;
[Deklaracja stałych]	<b>CONST</b> PI=3.14159265;
[Deklaracja typów danych]	<b>TYPE</b> TMOj = 1..10;
[Deklaracja zmiennych]	<b>VAR</b> X : TMOj;
	<b>FUNCTION</b> kwadrat(X : Integer):Integer;
	<b>BEGIN</b>
[Treść własnych procedur i funkcji]	Kwadrat:=X*X;
	<b>END</b> ;
<b>BEGIN</b>	<b>BEGIN</b>
[Treść programu]	X:=4;
	Writeln(Kwadrat(X));
<b>END.</b>	<b>END.</b>

## Przydatne procedury i funkcje

Poniżej przedstawiony został skrót przydatnych procedur i funkcji. Więcej informacji na temat każdej z nich, a przede wszystkim informacje, w jakim module się znajdują i przykłady zastosowania znaleźć można w systemie pomocy Turbo Pascala.

- Write, Writeln – wyświetlanie napisów na ekranie
- Read, Readln – pobieranie danych od użytkownika, np.:

```
VAR x : Integer;
...
Read(X);
```

Odczyt procedury Read kończy się po napotkaniu dowolnego białego znaku, a więc np. spacji, tabulatora, czy przejścia do nowej linii. Procedura ReadLn czyta ciąg z klawiatury aż do napotkania znaku końca linii. Pobrany ciąg znaków (lub liczba) zapamiętywany jest w zmiennej podanej jako argument.

- ClrScr; - czyszczenie ekranu
- TextColor(kolor); - zmiana koloru liter (kolor – liczba od 0..15 lub stała, np.: BLUE, GREEN)
- TextBackGround(kolor); - zmiana koloru tła (podobnie jak TextColor, tylko zakres od 0..7)
- GotoXY(X, Y); - ustawienie kursora na ekranie (X od 1..80, Y od 1..25)
- SQRT(Liczba); - obliczenie pierwiastka kwadratowego
- COS(X), SIN(X), TAN(X); - obliczanie wartości funkcji trygonometrycznych
- CHR(liczba); - zwraca znak ASCII o danym numerze
- ORD(znak); - zwraca kod ASCII danego znaku
- Length(Napis); - zwraca długość danego napisu
- Copy(Napis, Początek, Długość); - zwraca fragment tekstu przekazanego w zmiennej Napis, od pozycji Początek przez następne Długość znaków.
- Delete(Napis, Początek, Długość); - usuwa Długość znaków od pozycji Początek w zmiennej Napis.
- Str(Liczba, Napis); - zamienia liczbę na napis

- VAL (Napis, Liczba, Kod\_bledu) ; - zamienia napis na liczbę
- Delay (czas) ; - zatrzymanie wykonywania programu na czas milisekund
- ReadKey; - funkcja pobiera klawisz z klawiatury
- KeyPressed; - funkcja sprawdza, czy wciśnięto klawisz na klawiaturze
- Sound (HZ) ; - wydobywanie dźwięku o częstotliwości HZ
- NoSound; - przerwanie wydobywania dźwięku
- UpCase (Znak) ; - zamienia wielkość znaku na wielką
- Random (zakres) ; - wylosowanie liczmy z zakresu od 0..zakres-1. Przed użyciem (najlepiej na początku programu) należy wywołać procedurę Randomize;

## Przykładowy program

Poniżej znajduje się przykładowy program napisany w Turbo Pascalu wykorzystujący dynamiczną strukturę danych, mianowicie drzewo binarne. Dzięki jego analizie można utrwalić sobie wiadomości na temat struktury i składni w Pascalu, tworzeniu własnych procedury, rekurencyjnych typów danych, wykorzystaniu instrukcji iteracyjnych jak również dynamicznego operowania zmiennymi.

```
{BY BINBOY}
PROGRAM Drzewo;
TYPE
  TDrzewo = ^PDrzewo;
  PDrzewo = RECORD
    Liczba : Integer;
    L,P : TDrzewo;
  END;

VAR Korzen : TDrzewo;          {początek drzewa binarnego}
    L      : Integer;         {zmienna pomocnicza}

{***** DODAWANIE NOWEGO ELEMENTU DO DRZEWA *****}
PROCEDURE Dodaj(Liczba : Integer);
VAR Nowy : TDrzewo;
    Temp : TDrzewo;
    W    : Boolean;
BEGIN
  New(Nowy);
  Nowy^.L:=NIL;
  Nowy^.P:=NIL;
  Nowy^.Liczba:=Liczba;

  {Elementy większe na prawo, a mniejsze bądź równe na lewo}

  IF Korzen=NIL THEN Korzen:=Nowy
  ELSE BEGIN
    Temp:=Korzen;
    WHILE Temp<>NIL DO
      BEGIN
        W:=Liczba>Temp^.Liczba;
        IF W AND (Temp^.P=NIL) THEN BEGIN Temp^.P:=Nowy;Break;END;
        IF NOT W AND (Temp^.L=NIL) THEN BEGIN Temp^.L:=Nowy;Break;END;
        IF W AND (Temp^.P<>NIL) THEN Temp:=Temp^.P ELSE Temp:=Temp^.L;
      END;
    END;
  END;
END;

{***** WYSZUKIWANIE W DRZEWIE *****}
PROCEDURE Szukaj(Liczba : Integer);
```

```
VAR Temp : TDrzewo;
BEGIN
  Temp:=Korzen;
  WHILE (Temp<>NIL) AND (Temp^.Liczba<>Liczba) DO
  BEGIN
    IF Liczba>Temp^.Liczba THEN Temp:=Temp^.P
    ELSE IF Liczba<=Temp^.Liczba THEN Temp:=Temp^.L;
  END;
  IF Temp<>NIL THEN Writeln('Znaleziono!')
  ELSE Writeln('NIE znaleziono!');
END;

{***** PROGRAM GŁÓWNY *****}
BEGIN
  Korzen:=NIL;
  Writeln('Wprowadź liczby. Liczba -1 kończy wprowadzanie danych.');
```

Write('Podaj liczbę >>');

ReadLn(L);

WHILE L<>-1 DO

BEGIN

Dodaj(L);

Write('Podaj liczbę >>');

ReadLn(L);

END;

Writeln('Wyszukiwanie...');

Write('Jakiej liczby szukać? >>');

ReadLn(L);

WHILE L<>-1 DO

BEGIN

Szukaj(L);

Write('Jakiej liczby szukać? >>');

ReadLn(L);

END;

END.

Więcej przykładowych programów napisanych w Turbo Pascalu można znaleźć na stronie <http://binboy.org>.

## Uwagi końcowe

Pascal jest językiem łatwym do opanowania, szczególnie dla osób, które programowały już w C. Zachęcamy bardzo do “wstukiwania” przykładowych programów zawartych w tym dokumencie i eksperymentowania. Na pewno po ich analizie Czytelnik nie będzie miał problemów z implementacją bardziej złożonych programów.

## Zadania

Na wykonanie zadania przewidziane są dwa spotkania. Maksymalna liczba punktów do zdobycia wynosi 10, z czego 4 punkty można uzyskać za zadania przedstawione w instrukcji do laboratorium.

**Lista zadań** (zaproponował doc. dr inż. Wiesław Porębski oraz dr inż. Michał Małafiejski):

1. Napisz kalkulator liczb zespolonych wykonujący operacje: +, -, \*, /, |·|.
2. Napisz program sprawdzający metodą 0-1 czy dany schemat logiczny jest tautologią (operatory logiczne  $\Rightarrow, \Leftrightarrow, \wedge, \vee, \neg$  oraz zmienne  $p, q, r, s, t$ ).
3. Zaimplementuj algorytm tłumaczenia wyrażenia w postaci ONP do postaci wyrażenia algebraicznego z nawiasami (*wskazówka*: wykorzystaj dynamiczną strukturę drzewa binarnego).
4. Wyznacz wartość  $\sqrt[3]{a}$  dla danego  $a$  z dokładnością do 6 miejsc po przecinku (*wskazówka*: wykorzystaj metodę Newtona:  $x_{n+1} = \frac{2}{3}x_n + \frac{a}{3x_n^2}$ ).
5. Wyznacz dowolny pierwiastek wielomianu  $x^5 + ax^4 - bx^2 + cx - d$  z dokładnością do 6 miejsc po przecinku (*wskazówka*: wykorzystaj metodę bisekcji: jeżeli  $f(x) \cdot f(y) < 0$  dla pewnych  $x$  oraz  $y$ , wybierz  $t$  równe  $x$  lub  $y$  tak, żeby  $f((x + y) / 2) \cdot f(t) < 0$ , następnie iteruj metodę aż do uzyskania odpowiedniej dokładności).
6. Zaimplementuj algorytm Strassena szybkiego mnożenia macierzy (zob. *Wstęp do algorytmów*, Cormen, Leiserson, Rivest).
7. Dokonaj konwersji liczby wymiernej (dowolnej długości) do liczby wymiernej o podstawie  $k$  (z przecinkiem).
8. Zaimplementuj kalkulator ułamków wykonujący dodawanie, mnożenie, potęgowanie oraz odejmowanie. Wyniki powinny być ułamekami nieskracalnymi.
9. Dla wczytanej macierzy sąsiedztwa grafu  $G$  wyznaczyć liczbę wierzchołków w najliczniejszej krawędziowo składowej spójności.
10. Dla wczytanej macierzy sąsiedztwa grafu  $G$  ustalić odległości pomiędzy każdą parą wierzchołków (*wskazówka*: wykorzystaj algorytm Dijkstry dla składowych spójności).
11. Sprawdź czy dana macierz binarna  $n \times n$  jest macierzą spójnego grafu dwudzielnego.
12. Dla danej sieci połączeń kolejowych (odległości pomiędzy przynajmniej 10 miastami) ustal długości najkrótszych połączeń pomiędzy dwoma wybranymi miastami. Należy założyć, że połączenia są jednokierunkowe.
13. W danym ciągu liczbowym znajdź najdłuższy niemalejący podciąg (niekoniecznie kolejnych liczb).
14. Wyznacz  $k$  miejsc po przecinku w rozwinięciu dziesiętnym liczby pi.
15. W danym słowie znajdź najdłuższe podśłowo, które jest symetryczne (czytane od lewej oraz od prawej jest identyczne, np. w *abcbabcab* jest to *cbabc*).

## Wymagania

Dane należy pobrać ze standardowego wejścia, wyniki należy przesłać na standardowe wyjście. Oceniane będą następujące rzeczy:

- (2 pkt) program kompiluje się bez błędów i ostrzeżeń
- (2 pkt) program działa poprawnie dla przygotowanych przez studenta kilku zestawu danych testowych
- (2 pkt) za styl programowania - strukturalizacja (+1), komentarze i przejrzysty kod (+1)

Punkty karne (decyduje prowadzący):

- użycie goto (-1 pkt)
- niezwalnianie pamięci w dynamicznych strukturach danych (-1 pkt)