Krzysztof Regulski
AGH, WIMiIP

# PASCAL – PĘTLE

## 1. Zadanie 1

- Napisz program obliczający pole trójkąta lub prostokąta w zależności od wyboru użytkownika. Program powinien najpierw poprosić użytkownika o dwie długości [w mm], a następnie zapytać, czy chce liczyć pole trójkąta czy prostokąta. W odpowiedzi powinien podać:

  ```
  Pole [trójkąta/prostokąta] {w zależności od
  wyboru użytkownika} wynosi: [wynik] mm2.
  ```

- Narysuj algorytm tego programu na kartce. Następnie napisz program w dwóch wersjach: program z wykorzystaniem instrukcji `IF..THEN`, a następnie w drugiej wersji: program korzysta z instrukcji `CASE..OF`

## 2. Pętle

- Pętle pozwalają wykonywać pewne sekwencje czynności wielokrotnie, aż (dopóki) spełniony zostanie warunek. W `Pascalu` zaimplementowano trzy rodzaje pętli (plus instrukcja `GOTO`). Każda z nich odrobinę się od siebie różni i nadaje się lepiej do różnych zadań, jednak na początku będzie Ci się wydawało, że się wzajemnie zastępują.

## 3. Pętla `REPEAT..UNTIL`

- Ta pętla pozwala wykonać szereg poleceń zanim sprawdzony zostanie warunek, co daje pewność, że **wykona się przynajmniej raz**.
- Nadaje się do sytuacji, w których **nie wiemy z góry ile** będzie koniecznych iteracji (np. wpisywanie hasła przez użytkownika).
- **Przykład**: Napisz program, który będzie prosił użytkownika o hasło. Hasło będzie stałą, załóżmy, że napisem „`pascal`”. Pętla ta zakończy się dopiero wtedy, gdy użytkownik poda prawidłowe hasło. Wtedy wyświetli się napis "Brawo..."

```
program podaj_haslo;
Const
  haslo='pascal';
Var
  wprowadz:String;
Begin
 REPEAT
    writeln(' ' :10000);
    Write('Podaj haslo : ');
    ReadLn(wprowadz);
 UNTIL wprowadz=haslo;
 WriteLn(chr(7));     {sygnał dźwiękowy}
 writeln('BRAWO! Odgadłeś hasło!');
 readln;
End.
```

## 4. Pętla `WHILE..DO`

- Tutaj warunek zostaje sprawdzony na początku, przeciwnie do pętli `Repeat..Until`.
- Pętla ta działa dopóki warunek w niej zawarty **jest spełniany**. Jeśli warunek nie jest spełniony to nie wystartuje ona w ogóle – inaczej niż w przypadkach pozostałych pętli.
- Stosujemy ją kiedy nie wiemy ile iteracji ma ona wykonać.
- **Przykład**: Napisz program, który zadaje użytkownikowi pytania tak długo, aż użytkownik nie popełni błędu.

```
program milionerzy;
uses crt;
var
  a,w:integer;
  wynik:boolean;
begin
  a:=1;
  w:=2;
  wynik:=TRUE;
  while wynik=TRUE do
  begin
  write('Ile to jest ',a,'+',a,'? ');
    readln(w);
    if w=a+a then wynik:=TRUE else wynik:=FALSE;
    inc(a);
  end;
  writeln('Popełniłeś w końcu błąd, HA, HA!');
  readln;
end.
```

## 5. Zadanie 2 (1pkt)

- Napisz program, który najpierw pyta użytkownika o liczbę dwucyfrową, tak długo, aż podana zostanie cyfra określona w stałej (np.11) (wykorzystaj pętlę `REPEAT`), a kiedy zostanie podana wskazana liczba rozpocznij przepytywanie z tabliczki mnożenia (1*1, 2*2, 3*3,..) tak długo, aż użytkownik popełni błąd (pętla `WHILE`) .

## 6. Pętla `FOR..TO..DO`

- Pętla ta najlepiej sprawdza się w sytuacjach, w których dokładnie jesteśmy w stanie określić ile iteracji powinno zostać wykonanych.
- Pętla jest wyposażona w zmienną sterującą, czyli licznik iteracji.
- Dla tej pętli kompilator automatycznie sprawdzi, czy wartości początek i koniec są w zakresie typu zmiennej, tzn. czy może je przyjąć – musi mieć ona typ całkowity, porządkowy (`BYTE, WORD, LONGINT, INTEGER`). Zmienna musi zostać zadeklarowana.
- Porównajmy dwa przykłady, pierwszy z użyciem pętli `REPEAT`:

```
Var
 t:word; {typ word: liczba całkowita o pojemności 2 bajtów}
BEGIN
  t:=0;
  repeat
    t:=t+1;
    writeln('Wypisałem to zdanie już ',t,' raz');
  until t=20;
  readln;
END.
```

- a drugi z wykorzystaniem FOR:

```
Var
 t:word; BEGIN
FOR t:=1 TO 20 DO writeln('Wypisałem to zdanie już ',t,' raz');
readln;
END.
```

- Prawda, że czytelniej?
- **Przykład:** Pamiętasz zadanie z zajęć: Wprowadzenie do Pascala (zadanie 7.3), w którym program miał dwukrotnie wydrukować wizytówkę na ekranie? Miał wyglądać mniej więcej tak (ten wypisuje jeden raz):

```
Program wizytowka;
Uses CRT;
Const
  linia = '+------------------------------+';
var
   imie,nazwisko,adres: string;
BEGIN
  clrscr;
  write( 'Podaj swoje imię : ' );
  readln( imie );
  write( 'Podaj swoje nazwisko : ' );
  readln( nazwisko );
  write( 'Podaj swóje adres : ' );
  readln( adres );
  clrscr;
  TextColor(0);
  TextBackGround(15);
  GotoXY(20,10);
  writeln(linia);
  GotoXY(25,11);
  writeln(imie, ' ', nazwisko);
  GotoXY(25,12);
  writeln(adres);
  GotoXY(20,13);
  writeln(linia);
  readln;
END.
```

- Załóżmy, że chcemy, aby program wydrukował na ekranie zadaną przez użytkownika liczbę razy taką samą wizytówkę. Należy zapytać użytkownika ile razy powtórzyć wydruk, a następnie tyle razy powtórzyć operację. Jesteśmy w stanie dokładnie określić ile razy ma zostać wykonany wydruk, dlatego najlepiej skorzystać z pętli FOR:

```
Program wizytowka_for;
Uses CRT;
Const
  linia = '+------------------------------+';
var
   imie,nazwisko,adres: string;
   licznik_stop, licznik: byte;
BEGIN
  clrscr;
  write( 'Podaj swoje imię : ' );
  readln( imie );
  write( 'Podaj swoje nazwisko : ' );
  readln( nazwisko );
  write( 'Podaj swóje adres : ' );
  readln( adres );
  write( 'Ile razy wydrukować wizytówkę? : ' );
  readln( licznik_stop );
  clrscr;
FOR licznik:=1 TO licznik_stop DO
  BEGIN
    TextColor(0);
    TextBackGround(15);
    GotoXY(20,3+4*licznik);
    writeln(linia);
    GotoXY(25,4+4*licznik);
    writeln(imie, ' ', nazwisko);
    GotoXY(25,5+4*licznik);
    writeln(adres);
    GotoXY(20,6+4*licznik);
    writeln(linia);
  END;
  readln;
END.
```

- Przyjrzyj się teraz zagnieżdżonej pętli, wypisującej na ekranie współrzędne macierzy:

```
Program macierz;
uses CRT;
var
  i,j:byte;
BEGIN
  clrscr;
FOR i:=1 TO 9 DO
  BEGIN
    TextColor(black);
    TextBackGround(lightgray);
    write(' ',i,' ');              {wypisujemy numery wierszy}
    FOR j:=1 TO 9 DO
      BEGIN
        TextColor(yellow);
        TextBackGround(red);
        write(' (',i,',',j,') ');  {wypisujemy współrzędne}
      END;
    writeln;           {co by się stało bez tego polecenia?}
  END;
  readln;
END.
```

- **Zadanie 3**: wprowadź zmiany do programu, tak aby na ekranie wypisana została tabliczka mnożenia.
- **Zadanie 4 (za 1 punkt)**: Narysuj algorytm na kartce, a następnie napisz program, który pyta użytkownika o dowolny wyraz, a następnie przepisuje go „wstecz” (odwrotna kolejność liter). **Podpowiedzi**:
  - użyj funkcji zliczającej znaki w łańcuchu `length(string)`
  - odwołaj się do kolejnych liter za pomocą konstrukcji `string[index]`

## 7. Pętla `GOTO`

- Omawiając pętle należy także wspomnieć o konstrukcji `GoTo`. Nie jest to właściwie pętla, ale może zostać ta instrukcja w ten sposób użyta. Polecenie to bowiem przenosi bezwarunkowo w inne miejsce programu, oznaczone wcześniej etykietą (kotwicą). **Instrukcja ta nie jest polecana**, ze względu na zaciemnienie konstrukcji programu, warto jednak ją znać, żeby lepiej rozumieć programy napisane przez innych programistów.
- Dla przykładu posłużymy się programem, gdzie użytkownik będzie musiał wpisać hasło (patrz punkt 3 – pętla `REPEAT`). Jeżeli nie wpisze poprawnego hasła, program powtarza zapytanie. Jeżeli wpisze, program się wyłącza. Można ten program zrobić oczywiście za pomocą innych pętli już ci znanych, ale użyjemy tu polecenia "`GOTO`" - nie mylić z "`gotoXY`".
- Kotwica, czyli etykieta znakująca miejsce w programie, do którego mamy zamiar się przenieść, tak jak i zmienne - musi być wcześniej zdeklarowana. Zmienne deklarowaliśmy poleceniem "`VAR`", kotwice natomiast deklaruje się poleceniem "`LABEL`" i nie wpisuje się żadnych typów (po prostu je wymieniamy po przecinku bez żadnych dwukropków). Gotowy już program z hasłem, wygląda następująco:

```
program podaj_haslo_2;
Const
  haslo='pascal';
Var
  wprowadz:String;
LABEL miejsce1;
BEGIN
  miejsce1: {zwróć uwagę, że po kotwicy wytępuje dwukropek, a nie średnik!}
  writeln(' ' :10000);
  Write('Podaj haslo : ');
  ReadLn(wprowadz);
  IF wprowadz<>haslo THEN GOTO miejsce1;
  WriteLn(chr(7));     {sygnał dźwiękowy}
  writeln('BRAWO! Odgadłeś hasło!');
  readln;
End.
```

- W powyższym przykładzie kotwica o nazwie "`miejsce1`" znajduje się na początku programu. Kotwice mogą znajdować się również w innych miejscach. Kotwice wstawia się do programu tylko jeden raz. Wstawiamy jej nazwę i dwukropek.
- **Zadanie 5 (1pkt):** Zastosuj GOTO w programie, który pyta użytkownika, czy chce zakończyć działanie programu. Jeśli użytkownik wciśnie literę `T` (lub `t`), zakończy natychmiast, jeśli literę `N` (lub `n`), odczeka 3000ms i zapyta ponownie czy chce zakończyć jego działanie.