Kurs języka Object/Delphi Pascal

na bazie implementacji Free Pascal.

AUTOR ŁUKASZ STAFINIAK

Email: lukstafi@gmail.com, lukstafi@ii.uni.wroc.pl
Web: www.ii.uni.wroc.pl/~lukstafi

Jeśli zauważysz błędy na slajdach, proszę daj znać!

Wykład 2: Dane i instrukcje

Pierwotne typy danych i manipulacja nimi; składanie danych razem: rekordy, tablice.

Diagramy składniowe

Do opisu składni Pascala tradycyjnie używa się graficznej reprezentacji notacji EBNF. [patrz *Language reference guide*]

Pliki źródłowe Pascala

- Jak w większości języków, w leksyce mamy słowa zastrzeżone (kluczowe), identyfikatory, operatory, znaki odstępu („białe znaki”), stałe, komentarze.
- Pascal jest *nieWRażliwy na wieLkość liTer.*
- Komentarze:

```
(* This is an old style comment *)
{ This is a Turbo Pascal comment }
// This is a Delphi comment. All is ignored till the end of the line.
```

- Komentarze można zagnieżdżać.
- Rozszerzenia dla plików źródłowych:

  .pas. Standardowe.

  .pp. Używane m.in. przez źródła Free Pascal Compiler (FPC) dla plików niekompatybilnych z Delphi, nie polecane.

  .lpr. Lazarus PRoject. Można używać dla programów, żeby odróżnić je od modułów.

Tryby Free Pascala i dyrektywy kompilatora

`{$mode fpc}`. Tryb domyślny. Nie polecany! W zasadzie jest to Turbo Pascal z drobnymi rozszerzeniami.

`{$mode objfpc}`. Tryb Object Pascal. Polecany, tego będziemy zawsze używać. (Z wiersza poleceń: `-Mobjfpc`.)

`{$mode delphi}`. Tryb zgodności z Delphi. Pomimo że Delphi było pionierem wprowadzania cech językowych z popularnych języków obiektowych (C++, Java) do Object Pascala, część konstrukcji Free Pascala pojawiła się w Delphi później i z inną składnią, lub wcale.

`{$modeswitch objectivec2}`. Pod-tryb „Objective-Pascal" naśladujący język Objective-C, do pracy z Mac OS.

`{$H+}`. Interpretuje `String` jako `AnsiString`, zamiast `ShortString`. Polecane. Oryginalnie stringi w Pascalu miały długość maksymalnie 255 (obecny `ShortString`). Ciągle można używać stringów o zadanej długości `String[N]` z $N \leqslant 255$. (Także: `{$LONGSTRINGS ON}`. Z wiersza poleceń: `-Sh`.)

`{$COPERATORS ON}`. Pozwól na operatory przypisania jak w C: `+=`, `-=`, `*=`, `/=`. (Z wiersza poleceń: `-Sc`.)

`{$define Symbol}`, `{$ifdef Symbol}`, `{$else}`, `{$endif}`. Kompilacja warunkowa. Inne dyrektywy: `{$ifndef Symbol}`, `{$if COND}`, `{$elseif COND}`. Żeby zdefiniować `Symbol` z linii poleceń, przekaż do `fpc` parametr `-dSymbol`.

`-Ciort`. Parametry `fpc` – generuj testy: `-Ci` wejścia/wyjścia, `-Co` przepełnienia, `-Cr` zakresów tablic i typów podzakresowych, `-Ct` stosu.

`-gl`. Parametr `fpc` włączający `lineinfo` – dodaje pozycje w pliku do „stacktrace".

`-viwn`. Parametr `fpc` – wyświetl „info, warnings and notes". (Zazwyczaj są już w pliku `fpc.cfg`.)

Domyślne argumenty dla `fpc` można umieścić w pliku `~/fpc.cfg`, pod Linuxem patrz też w pliku `/etc/fpc.cfg`.

- Różnice pomiędzy Free Pascalem (w trybie `objfpc`) i Delphi omówimy jak poznamy już większość języka. Wykład zakłada implementację Free Pascal 2.6.0.

Typy bazowe

Type	Range	Size in bytes
Byte	0 .. 255	1
Shortint	-128 .. 127	1
Smallint	-32768 .. 32767	2
Word	0 .. 65535	2
Integer	either smallint or longint	size 2 or 4
Cardinal	longword	4
Longint	-2147483648 .. 2147483647	4
Longword	0 .. 4294967295	4
Int64	-9223372036854775808 .. 9223372036854775807	8
QWord	0 .. 18446744073709551615	8

Typy całkowitoliczbowe

Czytamy o typach w *Free Pascal: Reference guide*.

Przegląd operatorów, instrukcji i procedur

Typy wbudowane	System	System	math

`Integer Real`
`Longint String`
`Char Boolean`
	`WriteLn`
	`ReadLn`
	`ParamStr`
	`ParamCount`
		`Sin Cos`
		`Ln pi Exp`
		`Sqrt Power`
		`Random Round`
			`min degtorad`
			`max norm`
			`sum randg`
			`tanh power`

Sterujące	Typ boolowski	Typ napisów	Przypisanie `:=`

`if...then...else...`
`for...:=...to...do...`
`while...do...`
`repeat...until...`
`for...in...do...`
	`true false`
	`and or`
	`not xor`
		`+ 'Ala'`
		`Length`
		`SetLength`
		`Str`
		`Val`
			Zbiory: `><`
			`+ - * include`
			`in <= exclude`
			Dzielenie całk. `div`
			Modulo `mod`

`A : Array[1..10] of String;` `B : Array of Integer;`		`s[i] := '!';`	`^` – „spod wskaź.” `@` – pobierz adres
`SetLength (B, 1000);`		`**` tylko dostępny do przeciążania	

- Czytamy *Reference guide*: typy,wyrażenia i instrukcje
- Czytamy *Run Time Library*: moduły System i math

Przykłady programów

```
program RulerProg;
var ruler : String = '1';
begin
   ruler := ruler + ' 2 ' + ruler;
   ruler := ruler + ' 3 ' + ruler;
   ruler := ruler + ' 4 ' + ruler;
   WriteLn (ruler)
end.
```

```
program IntOps;
var
   a, b : Integer;
   sum, prod, quot, rem : Integer;
   ParseCodeA, ParseCodeB : Word;
begin
   Val (ParamStr (1), a, ParseCodeA);
   Val (ParamStr (2), b, ParseCodeB);
   if ParseCodeA + ParseCodeB = 0 then
   begin
      sum := a + b;
      prod := a * b;
      quot := a div b;
      rem := a mod b;
      WriteLn (a, ' + ', b, ' = ', sum);
      WriteLn (a, ' * ', b, ' = ', prod);
      WriteLn (a, ' / ', b, ' = ', quot);
      WriteLn (a, ' % ', b, ' = ', rem)
   end else begin
      WriteLn ('Error in a? Character ', ParamStr(1)[ParseCodeA]);
      WriteLn ('Error in b? Character ', ParamStr(2)[ParseCodeB])
   end
end.
```

```
program Quadratic;
var
   b, c, d, root1, root2 : Real;
   ParseCode : Word;
begin
   // parse coefficients from command line
   Val (ParamStr (1), b, ParseCode); if ParseCode <> 0 then exit;
   Val (ParamStr (2), c, ParseCode); if ParseCode <> 0 then exit;
   // calculate roots
   d := b*b - 4*c;
   if d >= 0 then begin
      d := sqrt (d);
      root1 := (-b + d) / 2;
      root2 := (-b - d) / 2;
      WriteLn ('root1 = ', root1 : 10:2);
      WriteLn ('root2 = ', root2 : 10:2)
   end
   else WriteLn ('determinant = ', d : 2:2)
end.
```

```
program LeapYear;
var
   Year : Integer;
   IsLeapYear : Boolean;
   ParseCode : Word;
begin
   Val (ParamStr (1), Year, ParseCode);
   if ParseCode <> 0 then exit;

   IsLeapYear := (Year mod 4 = 0) and (Year mod 100 <> 0);

   IsLeapYear := IsLeapYear or (Year mod 400 = 0);

   WriteLn (IsLeapYear)
end.
```

```
program SquareRoot;
const
   epsilon : Real = 1e-15;
var
   c, t : Real;
   ParseCode : Word;
begin
   Val (ParamStr (1), c, ParseCode);
   if ParseCode <> 0 then exit;

   t := c;
   while abs (t - c/t) > t*epsilon do
      t := (c/t + t) / 2;
   WriteLn (t : 0:2)
end.
```

Stała otypowana – mogą być dowolnego typu (ale muszą być „statycznie stałe”).

```
program Gambler;
var
   stake, goal, T, wins, experiment, cash : Integer;
   ParseCode : Word;
begin
   Val (ParamStr (1), stake, ParseCode); if ParseCode <> 0 then exit;
   Val (ParamStr (2), goal, ParseCode); if ParseCode <> 0 then exit;
   Val (ParamStr (3), T, ParseCode); if ParseCode <> 0 then exit;
   wins := 0;
   for experiment := 1 to T do
   begin
      cash := stake;
      while (cash > 0) and (cash < goal) do
         if random < 0.5 then Inc (cash)
         else Dec (cash);
      if cash = goal then Inc (wins)
   end;
   WriteLn (wins, ' wins of ', T)
end.
```

Musieliśmy zadeklarować cash od razu.

Random bez argumentu daje Real $\geqslant 0$ i <1, random(7) dałoby Integer w 0..6.

Inc i Dec zamiast C++/--.

```
program ShuffleDeck;
type
   Suit = (Clubs := 0, Diamonds, Hearts, Spades);
   LowRank = 2..10;
   Court = (Jack := High (LowRank)+1, Queen, King, Ace);
   Card = record
      suit : Suit;
      case IsFace : Boolean of
         True: (face : Court);
         False: (rank : LowRank)
   end;
const
   SuitNames : array[Clubs..Spades] of String
      = ('Trefl', 'Karo', 'Kier', 'Pik');
   FaceNames : array[Jack..Ace] of String
      = ('Walet', 'Dama', 'Król', 'As');
   N =
      (Ord (High (Court)) - Ord (Low (LowRank)) + 1) * 4 - 1;
var
   Deck : array[0..N] of Card;
   r, i, pos : Integer;
   s : Suit;
   c : Card;
```

Napisany „pod górkę", żeby pokazać różne konstrukcje typów wyliczeniowych, podzakresów, rekordów.

Wyliczenie i tak zacząłby się od zera.

Przesuwamy początek żeby zakresy `Ord` były rozłączne.

Pole rekordu po którym poznamy którego zestawu pól używać.

Rekord używający `case` będę nazywać *rekordem z wariantami*.

Stała tablica indeksowana typem wyliczeniowym.

`Ord` zwraca liczbę porządkową wartości wyliczeniowej.

Gdy nie ma powodu zaczynać od 1 to zaczynamy od 0 (bo tablice dynamiczne zawsze zaczynają się od 0).

Wszystkie rekordy tablicy `Deck` i rekord zmiennej `c` są od razu przydzielone na stosie.

```
begin
   for s in Suit do
      for r in [Low (LowRank)..Ord (High (Court))] do
      begin
         with Deck[(r - Low (LowRank))*4 + Ord (s)] do
         begin
            suit := s; IsFace := (r >= Ord (Low (Court))); rank := r
         end
      end;
   for i := 0 to N do
   begin
      pos := i + random (N-i+1);
      c := Deck[pos];
      Deck[pos] := Deck[i];
      Deck[i] := c
   end;
   for i := 0 to N do
      if ParamStr(1) = 'pl' then begin
         if Deck[i].IsFace then
            WriteLn (FaceNames[Deck[i].face], ' ', SuitNames[Deck[i].suit])
         else WriteLn (Deck[i].rank, ' ', SuitNames[Deck[i].suit]);
      end else begin
         if Deck[i].IsFace then
            WriteLn (Deck[i].face, ' of ', Deck[i].suit)
         else WriteLn (Deck[i].rank, ' of ', Deck[i].suit);
      end
end.
```

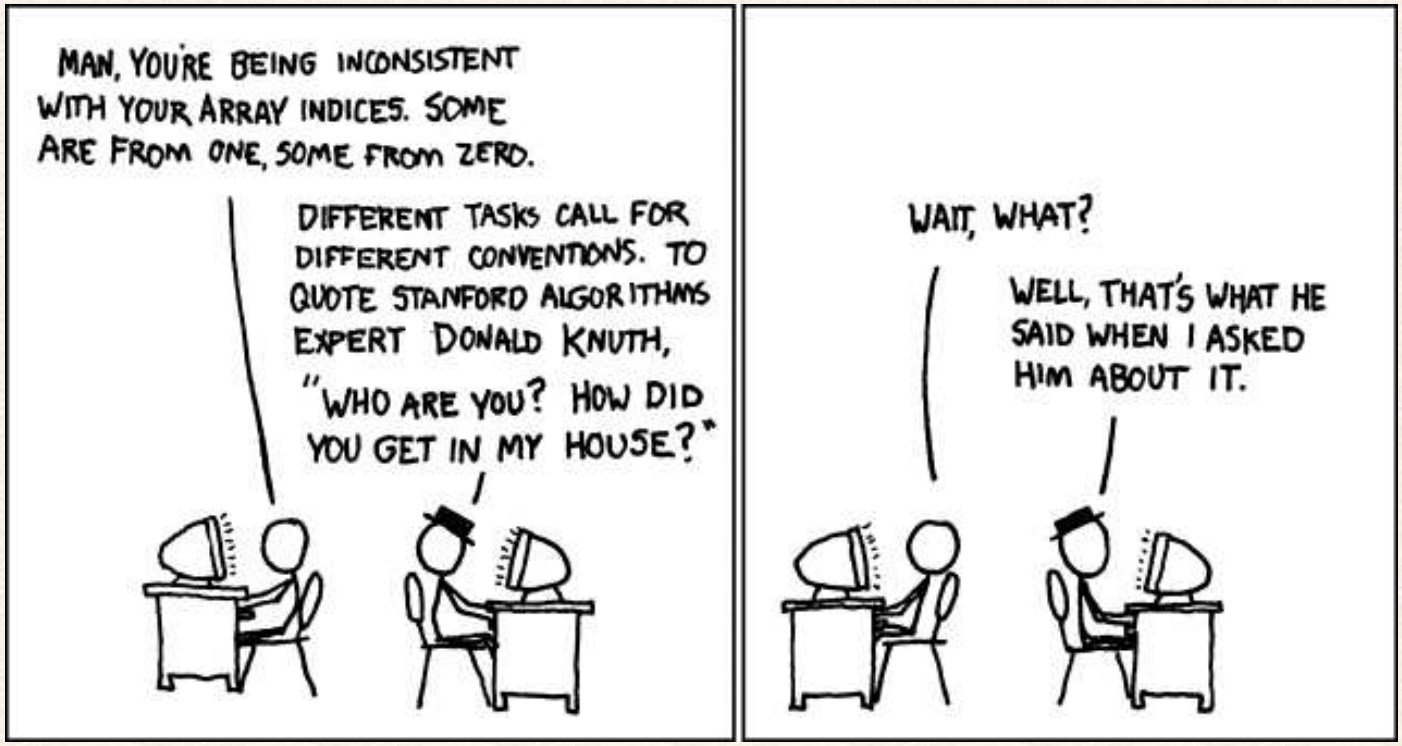
Pętla po typie wyliczeniowym.

Pętla po zbiorze wartości.

Zapisujemy pola rekordu.

Dzięki zapewnionej zgodności reprezentacji zapisujemy za jednym zamachem `rank` i `face`.

Przypadki rekordu z wariantami. Zazwyczaj użylibyśmy instrukcji `case...of...` ale tu warianty są po typie boolowskim.



http://imgs.xkcd.com/comics/donald_knuth.png

```
program MatrixMult;
{$COPERATORS ON}
type TMatrix = array of array of Real;
var
   M, N : Integer; ParseCode : Word;
   A,B,C : TMatrix;
   i,j,k : Integer;
begin
   Val (ParamStr(1), N, ParseCode); if ParseCode <> 0 then exit;
   Val (ParamStr(2), M, ParseCode); if ParseCode <> 0 then exit;
   SetLength (A, M, N);
   SetLength (B, N, M);
   SetLength (C, M, M);

   {code to fill A and B here}
   for i := 0 to M-1 do
      for j := 0 to M-1 do
         begin
            C[i,j] := 0;
            for k := 0 to N-1 do
               C[i,j] += a[i,k] * b[k,j];
         end
end.
```

```
program SelfAvoidingWalk;
var
   N, T : Integer;
   DeadEnds : Integer;
   walk, x, y : Integer;
   A : array of array of Boolean;
   ParseCode : Word;
   DeadEnd : Boolean;
   r : Real;
begin
   Val (ParamStr (1), N, ParseCode); if ParseCode <> 0 then exit;
   Val (ParamStr (2), T, ParseCode); if ParseCode <> 0 then exit;
   SetLength (A, N, N);
   DeadEnds := 0;
   for walk := 1 to T do
   begin
      for x := 0 to N-1 do
         for y := 0 to N-1 do
            A[x,y] := false;
      x := N div 2; y := N div 2;
      DeadEnd := false;
      while not DeadEnd
         and (x > 0) and (x < N-1) and (y > 0) and (y < N-1) do
      begin
         A[x,y] := true;
         {Check for dead end and make a random move.}
         if A[x-1,y] and A[x+1,y] and A[x,y-1] and A[x,y+1] then
         begin Inc (DeadEnds); DeadEnd := true
         end
```

Czyścimy planszę pod następny spacer.

Zamiast break używamy flagi kontrolującej opuszczenie pętli.

Porównania muszą być w nawiasach.

Kontynuacja na następnej stronie.

Pamiętaj że przy wypróbowywaniu kierunku tutaj nie zawsze robimy krok.

```
        else begin
           r := random;
           if      r < 0.25 then begin if not A[x+1,y] then Inc(x); end
           else if r < 0.50 then begin if not A[x-1,y] then Dec(x); end
           else if r < 0.75 then begin if not A[x,y+1] then Inc(y); end
           else if r < 1.00 then begin if not A[x,y-1] then Dec(y); end
        end
      end
   end;
   WriteLn ((100*DeadEnds) div T, '% dead ends')
end.
```