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 5: Pliki. Klasy generyczne.

Generyczne kontenery we Free Pascalu.

Pliki binarne i tekstowe

- Wygoda obsługi plików przyczyniła się do wczesnego sukcesu Pascala.
- `AssignFile` przypisuje nazwę pliku zmiennej plikowej.
- `Reset` otwiera plik do odczytu, `Rewrite` do zapisu od nowa, `Append` do dopisywania.
- Plik typu `file` to ciąg bajtów; nazywa się go plikiem beztypowym (ang. *untyped*). Czytamy z niego procedurą `BlockRead`, a piszemy procedurą `BlockWrite`.
- Pliki typów `file of` przechowują binarnie dane prawie dowolnego typu. Czytamy z nich procedurą `Read`, piszemy procedurą `Write`.
  - Nie przechowują danych zawierających obiekty klas ze zliczaniem dowiązań, dlatego nie użyjemy `AnsiString` (którego mamy domyślnie jako `String`), ale pochodzących ze starego Pascala stringów o ustalonej długości N, `String[N]`.

- Pliki `TextFile` zawierają tekst. Pisze się do nich przez `Write` i `WriteLn`, a czyta przez `Read` i `ReadLn` – potrafią one konwertować z typów bazowych. (Tak, to te same funkcje, standardowe wejście i wyjście to też `TextFile`!)
  - Kompilator zapisuje używając kodowania końca linii z systemu operacyjnego, a wczytując rozpoznaje różne kodowania (\n, \r, \r\n).
  - `Read` ze zmienną stringową czyta do końca linii, `ReadLn` dodatkowo zjada ten koniec.
  - Wczytując np. liczby nie trzeba korzystać z `ReadLn` –
- `AssignFile`, `CloseFile` i `TextFile` to oryginalnie `Assign`, `Close` i `Text`, ale nazwy `XxFile` są mniej mylące.
- Typy `file` / `file of` / `TextFile` określa się czasami „plikami starego typu". Za „nowoczesne" podejście do plików uważa się klasy `TStream` (o których za kilka wykładów).

Pliki: przykłady

```
program UntypedFileWrite;
uses SysUtils;
var
   MyFile: file;                                    Plik beztypowy.
   Data: array [0..99] of Byte;
   i : Integer;
begin
   for i := Low (Data) to High (Data) do            Przebiegamy po tablicy.
      Data[i] := Byte (i+30);
   AssignFile(MyFile, 'test.raw');
   Rewrite(MyFile, 1 { size of chunk });            Piszemy 1-bajtowe bloki.
   try
      BlockWrite(MyFile, Data, SizeOf(Data));       Ostatni argument
   finally                                          mówi o ilości bloków.
      CloseFile(MyFile);
   end;
end.
```

```
program UntypedFileRead;
uses SysUtils;
var
  MyFile: file;
  Data: array [0..99] of Byte;
  i : Integer;
begin
  AssignFile(MyFile, 'test.raw');
  Reset(MyFile, 1 { size of chunk });
  try
    BlockRead(MyFile, Data, SizeOf(Data));
    Write ('test.raw: ');
    for i := Low (Data) to High (Data) do
       Write (Char (Data[i]));
    WriteLn;
  finally
    CloseFile(MyFile);
  end;
end.
```

OK, bo `SizeOf` podaje rozmiar w bajtach.

```
program TypedFileWrite;
uses SysUtils;
type
   TPerson = record
      Name : String[80];
      Age : Integer;
   end;
var
  MyFile: file of TPerson;
  Person1: TPerson = (Name: 'Alice'; Age: 24);
  Person2: TPerson = (Name: 'Bob'; Age: 27);
begin
  AssignFile(MyFile, 'persons.raw');
  Rewrite(MyFile);
  try
    Write(MyFile, Person1, Person2);
  finally
    CloseFile(MyFile);
  end;
end.
```

String rozmiaru max. 80 znaków.

Zapisujemy dwa rekordy, binarnie.

```
program TypedFileRead;
uses SysUtils;
type
   TPerson = record
      Name : String[80];
      Age : Integer;
   end;
var
  MyFile: file of TPerson;
  Person1, Person2: TPerson;
begin
  AssignFile(MyFile, 'persons.raw');
  Reset(MyFile);
  try
    Read(MyFile, Person1, Person2);
    WriteLn ('persons.raw: ', Person1.Name, ' age ', Person1.Age,
             '; ', Person2.Name, ' age ', Person2.Age);
  finally
    CloseFile(MyFile);
  end;
end.
```

Typ odczytywany musi pokrywać się z zapisywanym.

```
program TextFileWrite;
uses SysUtils;
var
   MyFile : TextFile;
   Data: String = 'Żółcień żołędzi';
begin
   AssignFile(MyFile, 'test.txt');
   Rewrite(MyFile);
   try
      WriteLn(MyFile, Data);
   finally
      CloseFile(MyFile);
   end;
end.
```

Tekst będzie w Unicode.

```
program TextFileRead;
uses SysUtils;
var
   MyFile : TextFile;
   Data: String;
begin
   AssignFile(MyFile, 'test.txt');
   Reset(MyFile);
   try
      Read(MyFile, Data);
      WriteLn ('test.txt: ', Data);
   finally
      CloseFile(MyFile);
   end;
end.
```

```
program TextFileWriteConv;
uses SysUtils;
var
   MyFile : TextFile;
   DataS: String = 'DrzwiRazyPi';
   DataI: Integer = -2;
   DataF: Double = -3.14;
begin
   AssignFile(MyFile, 'test-conv.txt');
   Rewrite(MyFile);
   try
      WriteLn (MyFile, DataI, ' ', DataF);
      WriteLn (MyFile, DataS);
   finally
      CloseFile(MyFile);
   end;
end.
```

Gdyby zapisać `WriteLn (MyFile, DataS, ' ', DataI, ' ', DataF);` to byłby kłopot z wczytaniem – string „zjadłby" resztę tekstu. `ReadLn` nie jest konieczne jeśli wczytujemy np. tylko liczby (`Read` sobie poradzi).

```
program TextFileReadConv;
uses SysUtils;
var
   MyFile : TextFile;
   DataS: String;
   DataI: Integer;
   DataF: Double;
begin
   AssignFile(MyFile, 'test-conv.txt');
   Reset(MyFile);
   try
      ReadLn (MyFile, DataI, DataF);
      ReadLn (MyFile, DataS);
      WriteLn ('test-conv.txt: ', DataS, ' ', DataI, ' ', DataF);
   finally
      CloseFile(MyFile);
   end;
end.
```

Klasy generyczne

- Typy generyczne, wzorowane na szablonach z C++ (a nie na typach parametrycznych z Javy i C#), pojawiły się we Free Pascalu wcześniej niż w Delphi (ale rozwijane wolniej) stąd różnica w składni.
- Parametry generyczne zastępują typy.

  `generic TGTree<T,...> = class` deklaracja klasy z typami `T`, ...
- Implementacja metod przebiega podobnie jak dla zwykłych klas, poniżej deklaracji – ale mamy dostęp do parametrów `T` jako typów.
- Aby użyć typu generycznego, musimy go wyspecjalizować dla odpowiednich (nie-generycznych) typów-parametrów.

  ```
  type TIntList = specialize TFPGList<Integer>;
  ```

  Począwszy od FPC 2.6.0 zmniejszono konieczność deklarowania typów przed ich użyciem:

  ```
  var FooInt: specialize TFoo<Integer>;
  ```

- Począwszy od FPC 2.6, można też definiować generyczne tablice, rekordy (pewne ograniczenia dla rekordów z wariantami), funkcje, wskaźniki na metody.
- Typy generyczne w Delphi (wprowadzone dopiero w 2009): nie ma słów kluczowych `generic` i `specialize`, nie trzeba deklarować wyspecjalizowanego typu. Począwszy od FPC 2.6, tryb `{$mode Delphi}` stara się być kompatybilny z tymi możliwościami.
- Klasa generyczna widzi tylko nazwy z kontekstu swojej deklaracji, czyli przeciążenia procedur / operatorów widoczne w czasie specjalizacji nie zostaną uwzględnione.
- Niestety kontenery generyczne `TFPGList` / `TFPGMap` nie radzą sobie obecnie (m.in.) z rekordami.
  - Problem jest właśnie w tym że potrzebują operatora `=`.
  - Możliwe rozwiązanie: moduł `GenericStructList` ze źródeł Vrml Game Engine https://vrmlengine.svn.sourceforge.net/svnroot/vrmlengine/trunk/kambi_vrml_game_engine/src/base/genericstructlist.pas

```
program tgeneric25;
{$mode objfpc}{$H+}
type
  generic TArr<T> = array[0..2] of T;
  generic TDynamicArr<T> = array of T;
  generic TRecArr<T> = array[0..1] of record
    A: T; B: String;
  end;
var
  ArrInt: specialize TArr<Integer>;
  ArrStr: specialize TArr<String>;
  DynArrInt: specialize TDynamicArr<Integer>;
  RecArr: specialize TRecArr<Integer>;
begin
  ArrInt[0] := 1; ArrStr[0] := '1';
  SetLength(DynArrInt, 1);
  DynArrInt[0] := 2;
  RecArr[0].A := 3; RecArr[0].B := '3';
  if ArrInt[0] <> 1 then halt(1);
  if ArrStr[0] <> '1' then halt(2);
  if DynArrInt[0] <> 2 then halt(3);
  if RecArr[0].A <> 3 then halt(4);
end.
```

Przykłady z zastosowaniem kontenerów generycznych

```
program GenericListTest;
uses SysUtils, FGL;
type
   TPerson = class
      Name : String;
      Age : Integer;
      constructor Create (aName : String; anAge : Integer);
   end;
   TPersonList = specialize TFPGList<TPerson>;
   constructor TPerson.Create (aName : String; anAge : Integer);
   begin
      Name := aName; Age := anAge
   end;
```

```
var
   MyList: TPersonList;
   Person: TPerson;
begin
   MyList := nil;
   try
      MyList := TPersonList.Create;
      MyList.Add ('Alice', 24);
      MyList.Add ('Bob', 27);
      for Person in MyList do
         WriteLn ('person: ', Person.Name, ' age ', Person.Age);
   finally
      if MyList <> nil then
      begin
         for Person in MyList do
            Person.Free;
         MyList.Free
      end
   end
end.
```

```
program GenericMapTest;
uses SysUtils, FGL;
type TStrIntMap = specialize TFPGMap<String, Integer>;
var
   MyMap : TStrIntMap;
   pos : Integer;
   Name : String;
begin
   MyMap := nil;
   try
      MyMap := TStrIntMap.Create;
      MyMap.Add ('Zabini', 21);
      MyMap.Add ('Bob', 27);
      MyMap.Add ('Alice', 24);
      MyMap.Sort;
      Write ('Find person: '); ReadLn (Name);
      if MyMap.Find (Name, pos)
      then WriteLn ('Found ', Name, ': age ', MyMap.Data[pos])
      else WriteLn ('Not found ', Name)
   finally
      MyMap.Free
   end
end.
```

Bez posortowania nie znalazłby Alice!

```
program GenericMapTest2;
uses SysUtils, FGL;
type TStrIntMap = specialize TFPGMap<String, Integer>;
var
   MyMap : TStrIntMap;
   Name : String;
begin
   MyMap := nil;
   try
      MyMap := TStrIntMap.Create;
      MyMap.Add ('Zabini', 21);
      MyMap.Add ('Bob', 27);
      MyMap.Add ('Alice', 24);
      Write ('Find person: '); ReadLn (Name);
      try
         WriteLn ('Looking for ', Name, ': age... ', MyMap[Name])
      except on EListError do
         WriteLn ('Not found')
      end
   finally
      MyMap.Free
   end
end.
```

Nie posortowane więc MyMap[Name] w czasie liniowym.

Publiczne metody TFPGList i TFPGMap

```
generic TFPGList<T> = class(TFPSList)

constructor Create;
destructor Destroy; override;
function Add(const Item: T): Integer;
procedure Clear;
procedure Delete(Index: Integer);
procedure Exchange(Index1, Index2: Integer);
function Expand: TFPSList;
function Extract(const Item: T): T;
property First: T read GetFirst write SetFirst;
function GetEnumerator: TFPGListEnumeratorSpec;
function IndexOf(const Item: T): Integer;
procedure Insert(Index: Integer; const Item: T);
property Last: T read GetLast write SetLast;
procedure Assign(Source: TFPGList);
function Remove(const Item: T): Integer;
procedure Sort(Compare: TCompareFunc);
property Items[Index: Integer]: T read Get write Put; default;
property List: PTypeList read GetList;
procedure Move(CurIndex, NewIndex: Integer);
```

```
procedure Pack;
procedure Sort(Compare: TFPSListCompareFunc);
property Capacity: Integer read FCapacity write SetCapacity;
property Count: Integer read FCount write SetCount;
property ItemSize: Integer read FItemSize;
```

```
generic TFPGMap<TKey, TData> = class(TFPSMap)
constructor Create;
function Add(const AKey: TKey; const AData: TData): Integer;
function Add(const AKey: TKey): Integer;
function Find(const AKey: TKey; out Index: Integer): Boolean;
function IndexOf(const AKey: TKey): Integer;
function IndexOfData(const AData: TData): Integer;
procedure InsertKey(Index: Integer; const AKey: TKey);
procedure InsertKeyData(Index: Integer; const AKey: TKey; const AData:
TData);
function Remove(const AKey: TKey): Integer;
property Keys[Index: Integer]: TKey read GetKey write PutKey;
property Data[Index: Integer]: TData read GetData write PutData;
property KeyData[const AKey: TKey]:TData read GetKeyData write PutKeyData; default;
property OnKeyCompare: TKeyCompareFunc read FOnKeyCompare write SetOnKeyCompare;
property OnDataCompare: TDataCompareFunc read FOnDataCompare write SetOnDataCompare;
procedure Sort;
property Duplicates: TDuplicates read FDuplicates write FDuplicates;
property KeySize: Integer read FKeySize;
property DataSize: Integer read FDataSize;
property Sorted: Boolean read FSorted write SetSorted;
```

`TFPGList<T>`:

- Listy oparte są o automatycznie skalowane tablice (gdy brakuje miejsca, podwajające rozmiar dla małych i zwiększające o ponad 1/4 dla dużych; i kurczące się o połowę gdy wykorzystane poniżej 1/4 – ale dopiero dla dużych).
- `Add(const Item: T): Integer` wstawia element na koniec i zwraca jego pozycję. $O(1)$
- `Clear` usuwa wszystkie elementy.
- `Delete(Index: Integer)`usuwa element z listy. $O(n)$ ale kopiuje pamięć „naraz”.
- `Exchange(Index1, Index2: Integer)` – wiadomo.
- `Extract(const Item: T): T` zwraca element w liście równy argumentowi (według `IndexOf`) i usuwa go przez `Delete`.
- `First: T` zwraca lub nadpisuje element na pozycji 0 (początek).
- `Last: T` zwraca lub nadpisuje ostatni element (na pozycji o 1 mniejszej niż miałby kolejny dodany element).

- `GetEnumerator` pozwala używać składni `for...in`
- `IndexOf(const Item: T): Integer` liniowo szuka pierwszego elementu równego `Item`. „$O(n)$”.
- `Insert(Index: Integer; const Item: T)` wstawia element, kopiując „naraz” elementy na prawo od pozycji – $O(n)$.
- `Remove(const Item: T): Integer` zwraca pozycję elementu w liście równego argumentowi jednocześnie usuwając go przez `Delete`. $O(n)$
- `Sort(Compare: TCompareFunc)` quicksortuje.
- `Assign(Source: TFPGList)` czyści listę i kopiuje z innej element-po-elemencie – $O(n)$.
- `Items[Index: Integer]: T;default` pozwala używać listy jak tablicy.
- `Pack` usuwa elementy `nil` (lub 0...) z listy. $O(n)$.
- `Count` zwraca ilość elementów w liście, a `Capacity` przydzieloną pojemność listy.

`TFPGMap<TKey, TData>`:

- Wewnętrzna reprezentacja map jest taka sama jak list, na wzór list asocjacyjnych (klucz, wartość, klucz, wartość, ...)
- Dwa tryby operacji pomiędzy którymi można się przełączać: jako lista posortowana względem kluczy, wtedy znajdowanie jest w czasie logarytmicznym (binary search), albo nie posortowana, wtedy wstawianie jest w czasie stałym.
- `Add(const AKey: TKey; const AData: TData): Integer` wstawia asocjację klucz-wartość i zwraca jej pozycję. W trybie nie-posortowanym, wstawia na koniec, $O(1)$. W trybie posortowanym działa tylko jeśli klucz równy danemu już jest w liście – $O(\log n)$, i podmienia.
- `Add(const AKey: TKey): Integer` jak wyżej, ale ignoruje `AData`.
- `Find(const AKey: TKey; out Index: Integer): Boolean` szuka pozycję klucza w liście asocjacyjnej, i odpowiada czy znalazł. **Poprawnie działa tylko trybie posortowanym!** $O(\log n)$.
- `IndexOf(const AKey: TKey): Integer` zwraca pozycję podanego klucza, lub -1 jeśli nie ma go w liście. W trybie posortowanym $O(\log n)$, w nie-posortowanym $O(n)$.

- `IndexOfData(const AData: TData): Integer` zwraca pozycję asocjacji danej wartości. $O(n)$.
- `InsertKey(Index: Integer; const AKey: TKey)` nadpisuje klucz, a `InsertKeyData(Index: Integer; const AKey: TKey; const AData: TData)` klucz i wartość na danej pozycji. Tylko w trybie nieposortowanym. $O(1)$.
- `Remove(const AKey: TKey): Integer` znajduje klucz i usuwa jego pozycję (asocjacje powyżej się przesuną). $O(n)$.
- `Keys[Index: Integer]: TKey` i `Data[Index: Integer]: TData` pozwalają odnosić się do kluczy i wartości przez pozycję. $O(1)$.
- `KeyData[const AKey: TKey]:TData; default` pozwala traktować mapę jako tablicę indeksowaną typem `TKey`, z wartościami `TData`. Działa poprzez `IndexOf`, a przy przypisywaniu do nieistniejącego klucza, jako `Add`. Przy czytaniu z nieistniejącego klucza rzuca wyjątek `EListError`.
- `Sort` sortuje listę względem kluczy i przełącza do trybu posortowanego. `Sorted: Boolean` sprawdza czy w trybie posortowanym.
- `Duplicates,TDuplicates = (dupIgnore, dupAccept, dupError)` – czy dopuszczamy duplikaty kluczy (równe klucze na różnych pozycjach).

Listy zwalniające dane

Listy `generic TFPGObjectList<T>` ...

Moduł `StrUtils` – operacje na stringach

Tylko niektóre nieoczywiste funkcje opisałem. Patrz *Run Time Library: reference guide*.

- Niewrażliwe na wielkość liter szukaj / zastąp:

```
AnsiResemblesText(const AText, AOther: string): Boolean;
```
Czy takie same w wymowie American English.

```
AnsiContainsText(const AText, ASubText: string): Boolean;
AnsiStartsText(const ASubText, AText: string): Boolean;
AnsiEndsText(const ASubText, AText: string): Boolean;
AnsiReplaceText(const AText, AFromText, AToText: string): string;
AnsiMatchText(const AText: string; const AValues: array of string):
Boolean;
AnsiIndexText(const AText: string; const AValues: array of string):
Integer;
```
Funkcje `Match`/`Index` szukają stringa w tablicy.

- Zamieniając `XxText` na `XxStr` dostajemy warianty wrażliwe na wielkość liter.

- Różne:

```
DupeString(const AText: string; ACount: Integer): string;
```
Połącz ACount kopii AText.

```
ReverseString(const AText: string): string;
AnsiReverseString(const AText: AnsiString): AnsiString;
StuffString(const AText: string; AStart, ALength: Cardinal;  const ASubText:
string): string;
```
Zastąp wskazaną część stringa innym.

```
RandomFrom(const AValues: array of string): string; overload;
IfThen(AValue: Boolean; const ATrue: string; const AFalse: string = ''):
string; overload;
```

- Prefiks / postfiks stringa: (można opuścić przedrostek Ansi)

```
AnsiLeftStr(const AText: AnsiString; const ACount: Integer): AnsiString;
AnsiRightStr(const AText: AnsiString; const ACount: Integer): AnsiString;
AnsiMidStr(const AText: AnsiString; const AStart, ACount: Integer):
AnsiString;
```

- Różne:

```
IsEmptyStr(const S: string; const EmptyChars: TSysCharSet): Boolean;
DelSpace(const S: string): string;
```
Usuń wszystkie spacje.

```
DelChars(const S: string; Chr: Char): string;
DelSpace1(const S: string): string;
```
Zastąp ciągi spacji pojedynczą spacją.

```
Tab2Space(const S: string; Numb: Byte): string;
NPos(const C: string; S: string; N: Integer): Integer;
```
Pozycja N-tego wystąpienia stringa.

```
RPosEX(C:char;const S : AnsiString;offs:cardinal):Integer; overload;
RPosex (Const Substr : AnsiString; Const Source : AnsiString;offs:cardinal)
: Integer; overload;
RPos(c:char;const S : AnsiString):Integer; overload;
RPos (Const Substr : AnsiString; Const Source : AnsiString) : Integer;
overload;
```
Ostatnia pozycja wystąpienia w stringu źródłowym.

```
AddChar(C: Char; const S: string; N: Integer): string;
AddCharR(C: Char; const S: string; N: Integer): string;
```
Dodaj kopie znaku (R: na końcu) do uzyskania odpowiedniej długości wynikowego stringa.

```
PadLeft(const S: string; N: Integer): string;inline;
PadRight(const S: string; N: Integer): string;inline;
PadCenter(const S: string; Len: Integer): string;
```
Dodaj spacje do uzyskania odpowiedniej długości wynikowego stringa.

```
Copy2Symb(const S: string; Symb: Char): string;
Copy2SymbDel(var S: string; Symb: Char): string;
Copy2Space(const S: string): string;
Copy2SpaceDel(var S: string): string;
```
Skopiuj prefiks stringa (`Del` – kasując go ze stringa źródłowego).

```
WordCount(const S: string; const WordDelims: TSysCharSet): Integer;
```
Ilość słów w stringu.

```
WordPosition(const N: Integer; const S: string; const WordDelims:
TSysCharSet): Integer;
ExtractWord(N: Integer; const S: string;  const WordDelims: TSysCharSet):
string;
```
Skopiuj `N`-te słowo ze stringa. `StdWordDelims` można przekazać.

```
ExtractWordPos(N: Integer; const S: string; const WordDelims: TSysCharSet;
var Pos: Integer): string;
ExtractDelimited(N: Integer; const S: string;  const Delims: TSysCharSet):
string;
```
Zwraca `N`-ty string oddzielony `Delims`, może być pusty (np. parsowanie pól oddzielonych przecinkami lub średnikami).

```
ExtractSubstr(const S: string; var Pos: Integer;  const Delims:
TSysCharSet): string;
IsWordPresent(const W, S: string; const WordDelims: TSysCharSet): Boolean;
```

```
FindPart(const HelpWilds, InputStr: string): Integer;
IsWild(InputStr, Wilds: string; IgnoreCase: Boolean): Boolean;
GetCmdLineArg(const Switch: string; SwitchChars: TSysCharSet): string;
```
Pobierz argument z linii poleceń zaraz po argumencie zawierającym przełącznik.

```
IntToRoman(Value: Longint): string;
RomanToInt(const S: string): Longint;
const
  DigitChars = ['0'..'9'];
  Brackets = ['(',')','[',']','{','}'];
  StdWordDelims = [#0..' ',',','.',';','/','\',':','' '','"',''''] +
Brackets;
  StdSwitchChars = ['-','/'];
```

```
PosSet (const c:TSysCharSet;const s : ansistring ):Integer;
PosSet (const c:string;const s : ansistring ):Integer;
PosSetEx (const c:TSysCharSet;const s : ansistring;count:Integer
):Integer;
PosSetEx (const c:string;const s : ansistring;count:Integer ):Integer;
```
Znajdź pozycję dowolnego spośród podanych znaków.

```
Removeleadingchars(VAR S : AnsiString; Const CSet:TSysCharset);
RemoveTrailingChars(VAR S : AnsiString;Const CSet:TSysCharset);
RemovePadChars(VAR S : AnsiString;Const CSet:TSysCharset);
```

```
TrimLeftSet(const S: String;const CSet:TSysCharSet): String;
TrimRightSet(const S: String;const CSet:TSysCharSet): String;
TrimSet(const S: String;const CSet:TSysCharSet): String;
```

Wariant `Remove` modyfikuje stringa, wariant `Trim` zwraca nowy string.

Uwaga: `SysUtils` też zawiera operacje na stringach, np.:

`Trim`, `TrimLeft`, `TrimRight` – jak wyżej, ale tylko białe znaki.