Przeciążanie operatorów

1. Przeciążanie operatorów - teoria

W języku Delphi w ramach typów rekordowych można przeciążać pewne operatory czyli spowodować ich działanie inne niż standardowe.
Poszczególne operatory mają swoje nazwy, które są odwzorowywane w odpowiednie symbole np. nazwa *Substract* jest odwzorowywana w symblo - (minus).
W przypadku przeciążenia operatora kompilator wybiera odpowiedni sposób jego działania na podstawie kontekstu (typu wartości wyrażenia i typu parametrów użytych w wyrażeniu)
Przeciążanie operatorów pozwala na dokonywanie różnych działań na obiektach klas/rekordach np. mnożenie, dzielenie, dodawania etc., tak samo jak na zmiennych i stałych. Pozwala to na uproszczenie tworzonego kodu oraz zwiększa jego przejrzystość.
Np.

```
var
  A, B : TMyClass;
begin
  A := 20;
  B := A * A;
end;
```

Stosując powyżej operator mnożenia (*) w przypadku obiektów klas/rekordów, w rzeczywistości powodujemy wywołanie funkcji *Multiply* zdefiniowanej w ramach klasy/rekordu *TMyClass*.

Operatory, dla których dozwolone jest stosowanie przeciążeń:

Nazwa Operatora	Kategoria	Sygnatura deklaracyjna	Oznaczenie symboliczne
Implicit	Konwersja	Implicit(a : typ) : typ-wyniku;	niejawna zmiana typu
Explicit	Konwersja	Explicit(a: typ) : typ-wyniku;	jawna zmiana typu
Negative	Operator jednoargumentowy	Negative(a: typ) : typ-wyniku;	**-**
Positive	Operator jednoargumentowy	Positive(a: typ): typ-wyniku;	**+**
Inc	Operator jednoargumentowy	Inc(a: typ) : typ-wyniku;	Inc
Dec	Operator jednoargumentowy	Dec(a: typ): typ-wyniku	Dec
LogicalNot	Operator	LogicalNot(a: typ): typ-wyniku;	**not**

	jednoargumentowy		
Trunc	Operator jednoargumentowy	Trunc(a: typ): typ-wyniku;	Trunc
Round	Operator jednoargumentowy	Round(a: typ): typ-wyniku;	Round
In	Operator teoriomnogościowy	In(a: typ; b: typ) : Boolean;	**in**
Equal	Porównanie	Equal(a: typ; b: typ) : Boolean;	**=**
NotEqual	Porównanie	NotEqual(a: typ; b: typ): Boolean;	**<>**
GreaterThan	Porównanie	GreaterThan(a: typ; b: typ) Boolean;	**>**
GreaterThanOrEqual	Porównanie	GreaterThanOrEqual(a: typ; b: typ): Boolean;	**>=**
LessThan	Porównanie	LessThan(a: typ; b: typ): Boolean;	**<**
LessThanOrEqual	Porównanie	LessThanOrEqual(a: typ; b: typ): Boolean;	**<=**
Add	Operator dwuargumentowy	Add(a: typ; b: typ): typ-wyniku;	**+**
Subtract	Operator dwuargumentowy	Subtract(a: typ; b: typ) : typ-wyniku;	**-**
Multiply	Operator dwuargumentowy	Multiply(a: typ; b: typ) : typ-wyniku;	*****
Divide	Operator dwuargumentowy	Divide(a: typ; b: typ) : typ-wyniku;	**/**
IntDivide	Operator dwuargumentowy	IntDivide(a: typ; b: typ): typ-wyniku;	**div**
Modulus	Operator dwuargumentowy	Modulus(a: typ; b: typ): typ-wyniku;	**mod**
LeftShift	Operator	LeftShift(a: typ; b: typ): typ-wyniku;	**shl**

	dwuargumentowy		
RightShift	Operator dwuargumentowy	RightShift(a: typ; b: typ): typ-wyniku;	**shr**
LogicalAnd	Operator dwuargumentowy	LogicalAnd(a: typ; b: typ): typ-wyniku;	**and**
LogicalOr	Operator dwuargumentowy	LogicalOr(a: typ; b: typ): typ-wyniku;	**Or**
LogicalXor	Operator dwuargumentowy	LogicalXor(a: typ; b: typ): typ-wyniku;	**xor**
BitwiseAnd	Operator dwuargumentowy	BitwiseAnd(a: typ; b: typ): typ-wyniku;	**and**
BitwiseOr	Operator dwuargumentowy	BitwiseOr(a: typ; b: typ): typ-wyniku;	**Or**
BitwiseXor	Operator dwuargumentowy	BitwiseXor(a: typ; b: typ): typ-wyniku;	**xor**

Deklaracja operatora przeciążonego w typie rekordowym ma postać:

class operator *sygnatura deklaracyjna*

gdzie *sygnatura deklaracyjna* może mieć jedną z postaci podanych w powyższej tabeli.

Po definicji typu rekordowego z deklaracją operatora przeciążonego musi wystąpić deklaracja tego operatora (w tym samym programie, module lub bibliotece łączonej dynamicznie).
Postać definicji operatora przeciążonego jest podobna do definicji metody, tylko, że zamiast słów **procedure**, **function**, **constructor** lub **destructor** występują słowa **class operator**. Definicja taka powinna określać nową funkcjonalność danego operatora.

```
type
   typeName = record
       class operator conversionOp(a: type): resultType;
       class operator unaryOp(a: type): resultType;
       class operator comparisonOp(a: type; b: type): Boolean;
       class operator binaryOp(a: type; b: type): resultType;
   end;
```

Jeśli następnie zadeklarujemy zmienną (lub zmienne) danego typu rekordowego i w stosunku do tej zmiennej (lub zmiennych) posłużymy się oznaczeniem symbolicznym operatora (np. +), to zostanie wykonana operacja (może to być kilka operacji) określona w definicji.

Jeśli operator dwuargumentowy zostanie zdefiniowany dla danego typu rekordowego, to zostanie on użyty tylko w przypadku, gdy w wyrażeniu co najmniej jeden z argumentów będzie tego typu rekordowego.

Dla operatora jednoargumentowego zdefiniowanego w typie rekordowym albo typ argumentu, albo typ wyrażenia musi być typu rekordowego.

```
type
   TMyRecord = record
     // Addition of two operands of type TMyRecord
     class operator Add(a, b: TMyRecord): TMyRecord;
     // Subtraction of type TMyRecord
     class operator Subtract(a, b: TMyRecord): TMyRecord;
     // Implicit conversion of an Integer to type TMyRecord
     class operator Implicit(a: Integer): TMyRecord;
     // Implicit conversion of TMyRecordto Integer
     class operator Implicit(a: TMyRecord): Integer;
     // Explicit conversion of a Double to TMyRecord
     class operator Explicit(a: Double): TMyRecord;
   end;

// Example implementation of Add
class operator TMyRecord.Add(a, b: TMyRecord): TMyRecord;
begin
   // ...
end;

var
x, y: TMyRecord;
begin
   x := 12;      // Implicit conversion from an Integer
   y := x + x;   // Calls TMyRecord.Add(a, b: TMyRecord): TMyRecord
   b := b + 100; // Calls TMyRecord.Add(b, TMyRecord.Implicit(100))
end;
```

2. Przykład

Zdefiniujmy typ rekordowy *dod_mod5* z deklaracjami operatorów przeciążonych:

```
type
   dod_mod5 = record
                  var w : Integer;
                  class operator Add(x, y: dod_mod5):Integer;
                  class operator Implicit(x: Integer):dod_mod5;
```

```
                    class operator Explicit(x: dod_mod5):Integer;
   end;

// definicje operatorów przeciążonych muszą wystąpić po definicji typu rekordowego dod_mod5

// dodanie do pierwszego argumentu wartości drugiego argumentu modulo 5. Wynik operacji jest
// typu Integer
class operator dod_mod5.Add(x, y: dod_mod5):Integer;
begin
  Result := a.w + b.w mod 5;
end;

// Oznacza niejawną konwersję wartości typu Integer na typ dod_mod5, a dokładniej przypisanie
// tej wartości polu w rekordzie typu dod_mod5
class operator dod_mod5.Implicit(x: Integer):dod_mod5;
begin
  Result.w := a;
end;

// Określa jawną konwersję wartości typu rekordowego dod_mod5 na typ Integer i oznacza, że w
// przypadku zmiany typu, czyli zastosowaniu konstrukcji Integer(x), gdzie x oznacza rekord
// typu dod_mod5, odpowiednia wartość ma być pobrana z pola tego rekordu.
class operator dod_mod5.Explicit(x: dod_mod5):Integer;
begin
  Result := a.w;
end;

var
  x, y: dod_mod5;
  a, b: Integer;

begin
 a := 20;
 b := 12;
 Writeln(a + b mod 5); // na ekranie pojawi się 22
 x := a; // niejawna zmiana typu, równoważne x.w := a;
 y := b; // niejawna zmiana typu, równoważne y.w := b;
 Writeln(x + y); // na ekranie pojawi się 22
 // wykorzystanie operatora jawnej zmiany typu z dod_mod5 na Integer
 Writeln(Integer(x)); // na ekranie pojawi się 20
 Writeln(Integer(y)); // na ekranie pojawi się 12
end.
```

3. Uwagi

W pakiecie Embarcadero Delphi XE2 znajduje się przykładowy program *Win32OperatorOverload*, w którym przedstawiono zastosowanie operatorów przeciążonych do wykonywania operacji na liczbach zespolonych. Program ten wykorzystuje moduł o

nazwie *VassBotn.Vcl.Complex*, w którym zdefiniowano działania na liczbach zespolonych i funkcje standardowe dla argumentów zespolonych.