# Decimal Implementation in IML

## Zwischenbericht, Ralf Grubenmann, Janis Peyer, Compilerbau, HS 2014, Team 06

## Beschreibung

Implementiert wird der Decimal Gleitkommazahl Datentyp, wie er in diversen Sprachen vorhanden ist. Diese Beschreibung stützt sich auf die C# Language Specification für den Typ decimal und entspricht dieser grösstenteils. Der Hauptunterschied und Vorteil von Decimal gegenüber Float ist, dass Zahlen in der Basis 10 repräsentiert werden. Dadurch lässt sich z.B. 0.1 exakt darstellen, was vorallem im Bankenwesen nützlich ist.

Decimal ist ein 128 Bit Datentyp, der vor allem für finanzielle Berechnungen nützlich ist. Er kann Werte von ungefähr $1.0 * 10^{-28}$ bis $7.9 * 10^{28}$ darstellen mit 28-29 Stellen Genauigkeit, der Wertebreich entspricht also $(-1)^s * c * 10^{-e}$ wobei s das Vorzeichen ist (0 oder 1), der Koeffizient c ist definiert als $0 \leq c < 2^{96}$ und der Exponent e als $0 \leq e \leq 28$.

Nicht unterstützt werden vorzeichenbehaftete Nullen (es gibt also nur +0 und kein -0), Unendlich oder NaN.

Bei Berechnungen mit Decimal wird jeweils zur nächsten validen Decimal Repräsentation gerundet. Wenn ein Wert gleich weit von 2 validen Repräsentationen entfernt ist, wird diejenige mit einer geraden Ziffer an hinterster Stelle gewählt (z.B. 0.00...0025 wird gerundet zu 0.00…002 und nicht 0.00….003), auch bekannt als Round half to even. Absolute Werte kleiner als $5 * 10^{-29}$ werden auf 0 gerundet, Werte die zu gross zur Darstellung sind ergeben einen Overflow-Fehler (In IML ohne Exception-Handling wäre das ein Crash), da besonders in finanziellen Applikationen jedes andere Verhalten zu inkorrekten Resultaten führt.

In arithmetischen Ausdrücken wird müssen beide Operanden Decimal sein oder, wenn nur ein Operand vom Typ Decimal ist, muss der andere ein Integer-Typ sein. In diesem Falle wird der Integer Typ automatisch zu Decimal gecastet. Es kann nach Int32 gecastet werden, in welchem Fall auf die nächste Integerrepäsentation gerundet wird (bei gleicher Distanz auf die nächste gerade Integer Repräsentation), falls der Wert den Wertbereich von Int32 übersteigt, gibt es einen Overflow Fehler.

Decimal Literale bestehend aus Ziffern, einem Dezimal-Trennzeichen (.) und dem Suffix „m“, z.b. 999.99m . Dies dient der zukünftigen Unterscheidung zu (noch nicht implementierten) float Werten.

Implementiert werden folgende Operatoren:

| Operator | Funktion | Präzedenz | Assoziativität |
|---|---|---|---|
| + | Addition | 2 | left |
| - | Subtraktion | 2 | left |
| * | Multiplikation | 3 | left |
| / | Division | 3 | left |
| = | Gleichheit | 1 | non |
| /= | Ungleicheit | 1 | non |
| < | Kleiner als | 1 | non |
| > | Grösser als | 1 | non |
| <= | Kleiner gleich | 1 | non |
| >= | Grösser gleich | 1 | non |

## Lexeme

Die lexikalische Analyse von IML hat sich mit unserer Änderung wie folgt verändert:

Zur Enumeration Type wurde das Element DECIMAL hinzugefügt. Dadurch können Identifier nun vom Typ decimal sein. Zusätzlich wurde eine neue Enumeration namens Casttype erstellt.

```
data Type = BOOL | INT32 | DECIMAL
data Casttype = INT32 | DECIMAL
```

Decimal Literale warden wie folgt erkannt:

| Pattern | Example Literal | Example Tokens |
|---|---|---|
| (+\|-)?[0-9]+(\.[0-9]+)?m | 1234.5678m | (LITERAL, DecimalVal 1234.5678)<br>(TYPE, Type DECIMAL)<br>(CASTTYPE, Casttype INT32) |

## Grammatik

Die Grammatik wurde wie folgt angepasst:

Für das nicht terminal Symbol <factor> wurde eine neue Produktion für das Casting erstellt.

```
<factor>
  LITERAL
  IDENT <optInitOrExprList>
  <monadicOpr> <factor>
  LPAREN <expr> RPAREN
  CASTTYPE LPAREN <expr> RPAREN
```

Falls Sie sich bei den Lexemen gefragt haben, weshalb Casttype eingefügt wurde, wird dies hier klar. Mit Casttype kann bereits zur Parsetime festgestellt werden, ob ein Casting grammatikalisch korrekt eingesetzt wird.

## Das Casting Problem

Zu Beginn war das Ziel das Casten syntaktisch mit Klammern um den Typ zu erstellen. Beispiel:

```
a := (int32) b
```

Diese Syntax wird in C-artigen Sprachen wie Java und C# verwendet. Es entsteht dadurch jedoch ein Problem mit der Erstellung der Parstabelle: Es darf mit dieser Syntax eine Casting Klammer an einer Position geschrieben werden, an welcher ebenfalls eine Expression Klammer auftreten darf.

Aufgrund dieser Probleme haben wir uns entschieden die folgende Casting Syntax zu verwenden:

```
a := int32(b)
```

## Beispiel IML Zinsrechnung

Die Formel für das Kapital nach n Jahren bei jährlicher Verzinsung und Zinseszinsen lautet:

$$K_n = K_0 \cdot (1+i)^n = K_0 \cdot q^n$$

Diese Formel wird im nachfolgenden IML Programm implementiert. Zusätzlich wird das Resultat noch auf 5 Rappen gerundet, um das casting zu zeigen.

```
program interest(in const capital0:decimal, in const interest:decimal,
     in const years:int32, out const capitaln:decimal)
global
     proc pow(in copy const base:decimal, in copy var exp:int32,
          out ref var o:decimal)
     do
          o init := 1.0m;
          while exp > 0 do
               o := o * base;
               exp := exp - 1
          endwhile
     endproc;
     var ipown:decimal; //pow(i, years)
     var captialur:decimal //Unrounded capital
do
     //Calculate interest
     call pow(1 + interest, years, ipown init);
     captialur := capital0 * ipown;
     //Round to 5 centime (Rappen)
     capitaln init := decimal(int32(captialur * 20.0m)) div 20.0m
endprogram
```