3. Variabler och datatyper

a. Variabeldeklaration

Varje parameter och variabel är av en viss datatyp. Funktioner som returnerar ett värde är också av en viss typ. Datatypen anges med ett kolon ':' efter namnet, så här.

```
[var/out/ ] namn : datatyp;
```

Variabelnamnet är "namn". Typen är "datatyp". En variabel kan beskrivas som en lagringsplats för ett värde. Lagringsplatsen har ett namn, variabelnamnet, för att man skall kunna referera till dess värde. För typerna integer, real, boolean och time finns dom motsvarande vektortyperna integervector, realvector, booleanvector och timevector. En vektor är i modellspråket en följd av tal (eller andra värden) av samma typ. Innehållet i (tal-)följden kallas element och åtkoms via [ ]-parenteser: "vektor[12]" motsvarar det 12:te elementet i vektorn.

Tidigare har vi som exempel definierat utvektor1 enligt:

```
out utvektor1 : realvector;
```

Där vi tidigare förklarat att 'out' innebär att datat ska användas som output i funktionen. Nu vet vi även att namnet som vektorn ska identifieras med är 'utvektor1' och datatypen är 'realvector'. Nu skall vi reda ut vilka olika datatyper som finns tillgängliga i Futurelook och hur dessa kan användas på olika sätt.

b. Nyckelord och namnkonventioner

Följande ord är reserverade för speciella ändamål i modellspråket kan inte användas som variabel- eller funktionsnamn.

- addgraphic
- addtoset
- and
- begin
- do
- else
- end
- exit
- false
- for
- getdata
- if
- len
- loop
- minutes
- not
- now
- or
- par
- removefromset
- repeat
- return
- then
- timevec
- to
- true
- undef
- until
- var
- while

c. Datatyper

i. Boolean, booleanvector

En variabel av datatypen boolean kan endast ha ett av två värden: true eller false. En booleanvector är således en vektor med endast boolska värden. Man anger värdet på en boolean genom := liksom alla andra variabler, men kan endast ange true eller false som värden. Anta att vi definierat b1 som boolean och b2 som en booleanvector enligt nedan:

```
var b1 : boolean; b2 : booleanvector; b1 := true;       // Sätter b1 till true b1 := false;     // Sätter b1 till false b2 := vektor1 > relati
```

ii. Instrument (kurser, volymer) även kallad objekt

`objekt.high`	Innehåller objektets högsta kurs under perioden (datatyp: realvector)
`objekt.low`	Innehåller objektets lägsta kurs under perioden (datatyp: realvector)
`objekt.open`	Innehåller objektets öppningskurs för perioden (datatyp: realvector)
`objekt.close`	Innehåller objektets stängningskurs för perioden (datatyp: realvector)
`objekt.vol`	Innehåller objektets volym för perioden (datatyp: realvector)
`objekt.evol`	Innehåller objektets efteranmälda volym för perioden (datatyp: realvector)
`objekt.code`	Innehåller objektets Vikingkod (datatyp: string)

Undervariablerna är av typer vektor eller string och samma regler gäller, t.ex.

`rv := objekt.close;`	Sätter vektorn rv till objektets close-vektor
`rv[i] := objekt.open[i];`	Sätter element i i rv-vektorn till element i objektets open-vektor
`str := objekt.code;`	Sätter strängvariabeln str till objektets code-sträng.

iii. Integer, integervector

En variabel av typen integer är ett heltal mellan -2147483646 och +2147483646. En integervector är en vektor där samtliga värden endast kan vara heltal. iv. Real, realvector

En variabel av typen real är ett decimaltal skriven med punkt (.) som decimaltecken och utan tusentalsseparator. Decimaltalen kan även beskrivas i grundpotensform x.xEy

`rv := objekt.close;`	Kopiera varje element i objekt.close till rv
`rv[i] := 2000.0;`	Sätt i:te elementet i rv till 2000.0
`rv[i] := 2.0E3;`	Sätt i:te elementet i rv till 2000.0 (i grundpotensform är 2000 = 2.0 * 10^3)
`rv[i] := 0.002;`	Sätt i:te elementet i rv till 0.002
`rv[i] := 2.0E-3;`	Sätt i:te elementet i rv till 0.002 (i grundpotensform är 0.002 = 2.0 * 10^-3)

v. Set (listor)

Set är datatypen som används för objektlistor i Futurelook. Set är ett antal objekt, som är representerade av vikingkoder. Set använder du, när du vill göra modeller där det läggs villkor på flera objekt. Följande funktioner är användbara vid hantering av set:

`RemoveFromSet(objektlista, 2);`	Ta bort det tredje elementet ur set-vektorn objektlista. Platsen lämnar inte tom utan platsen försvinner och alla element över 2 flyttas ner ett steg.
`AddToSet(objektlista, "190072");`	Lägger till objektet med Vikingkod 190072 sist i set-vektorn objektlista.

vi. String (text)

En sträng, eng. string, är en variabel som används för att spara en följd av tecken. Dessa tecken kan vara bokstäver, siffror och symboler och avgränsas av tecknen ” eller ’ . Strängar använder du t.ex. när du vill lägga in text i diagrammen eller som inparameter i funktioner.

# ¤ % &	Otillåtna tecken för string-variabeln
! / ( ) = ? @ £ $ € { [ ] } \ ~ , . - ' ¨ ´ + ; : _ * ^ ` § ½ < > |	Tillåtna tecken för string-variabeln

```
str := ”190072”;
```

eller

```
str := ’190072’;        // Strängen ”190072” sparas i string-variabeln str

AddToSet(objektlista, str);
```

Vikingkoden sparad i string-variabeln str adderas till set-variabeln objektlista.

```
Str := 'Vem är "Ben Bernanke"?';

Str := "Är Ben Bernanke en 'finansnörd' ?";
```

” och ’ kan användas parvis inuti varandra

vii. Time, timevector

Datatypen time är en variabel som representerar en tidpunkt. Det finns en mängd funktioner i funktionsbiblioteket std. Till skillnad från de andra datatyperna som vi gått igenom är inte timevector en vektor av enskilda element av time, utan en speciell vector som endast används i form av konstanten timevec i vissa funktioner För att ge en variabel av typen time ett visst värde använder man sig av funktionen

```
time(tidpunkt);
```

där tidpunkt är en tidsangivelse på korttidsformat (YYMMDD / YY-MM-DD / YYYY-MM-DD eller YYYYMMDD) eller långtidsformat (korttidsformat med efterföljande klockslag HH:MM:SS), enligt nedan:

```
tidpunkt1 := time("97-02-19"); tidpunkt1 := time("970219"); tidpunkt1 := time("1997-02-19"); tidpunkt1 := time("19970219");
```

Tidpunkt1 sätts till 19:e februari 1997 enligt korttidsformatet

```
tidpunkt2 := time("1992-09-01 12.20.30"); tidpunkt2 := time("19920901 12.20.30"); tidpunkt2 := time("92-09-01 12.20.30"); tidpunkt2 := time("9
```

Tidpunkt2 sätts till 1:a september 1992 kl 12:20:30 i alla dessa olika kombinationer av långtidsformatet

4. Satser, funktioner och konstanter

Kodens huvudbeståndsdelar är satser och uttryck. Uttryck kan inte stå ensamma utan måste finnas i en sats.
Satsen avslutas alltid med ett semikolon ;

```
i := (j + k) * l ;
```

Tilldelningssats bestående av satsen i := uttryck; och uttrycket (j+k)*l.
Resultatet av beräkningen (j+k) * l kommer att tilldelas variabeln i.

Funktioner som inte har ett returvärde kan anropas som en sats. Funktioner som har ett returvärde måste användas som ett uttryck eller del av ett uttryck. Här följer en beskrivning av alla satser, specialfunktioner (len och undef) och konstanter (now, minutes, timevec) som ingår i modellspråket. Listan är i bokstavsordning.

```
AddGraphic(graf,typ,text,x1,y1,x2,y2)
```

graf	Den var-definierade parametern som bestämmer i vilken panel som grafen ritas ut. Lämnas denna bort så ritas grafen ut i översta panelen
typ	Typ av grafik: Alternativen är: ”Line” och ”Text” inklusive ””-tecken
text	om typen är ”Text” ovan så ritas denna variabel ut i chartet. Måste vara av typen typen String, antingen variabel eller text inom ” ”-tecken
x1	Startposition i x-led. Integer som anger index I vektorn.
y1	Startposition i y-led. Real som anger vid vilken värde som grafiken ska börja ritas ut. Använder samma skala som variabeln graf ovan.
x2	Slutposition i x-led. Integer som anger index I vektorn.
y2	Slutposition i y-led. Real som anger vid vilken värde som grafiken ska sluta. Använder samma skala som variabeln graf ovan.

```
par(main : instrument; var rv : realvector); var begin AddGraphic(rv, "Line", "",now-100, main.close[now-100],now, main.close[now]); AddGraphi
```

Notera parametern rv i par-segmentet, denna är nödvändig för att använda som graf variabel oavsett om den anropas i AddGraphic(). I presentationsredigeraren har variabeln rv i detta fall lagts i den nedre rutan.

```
AddGraphic(rv, "Line", "",now-100, main.close[now-100],now, main.close[now]);
```

Den första AddGraphic() funktionen anväder sig av rv som grafvariabel och ritas därför ut i den nedre rutan. Vi definierar graftypen som ”Line”. Eftersom typen är Line så räcker det med att skriva ”” i text-delen. Grafen ska börja ritas ut 100 perioder innan senaste data (konstanten now) och sluta vid senate period och ska rita ut värdena vid dessa två tidpunkter. Den andra AddGraphic()-funktionen är nästan likadan, men använder sig inte av grafvariabeln rv, därför ritas den ut i den övre rutan. Vi ritar i detta fall ut en linje som är 50 perioder bak i tiden för att kunna skilja på dem. AddToSet lägger till vikingkodsträngar från en objektlista.

```
AddToSet(set, string);
```

set	En variable av datatyp set, d.v.s en objektlista
string	En sträng, antingen en variabel av datatyp sträng eller en textsträng mellan ” ”-tecken
AddToSet(list1, ”190072”);	Lägg till objektet OMX Stockholm 30 i objektlistan list1

For-satsen, eller for-loopen används när man vill utföra en kodsekvens för varje element i en vektor. For-loopen tar tre parametrar, den första kallas loopvariabel och är av typen integer. Variabelns värde ökas med ett för varje varv i loopen. Dom andra två parametrar är start- och slutvärde för loopvariabeln.

```
for loopvariabel := startvärde to slutvärde do //kod som ska utföras; end;
```

loopvariabel	En variabel av typen integer. Detta heltal räknar upp med +1 för varje gång loopen körs.
startvärde	Ett heltalsvärde som ges åt loopvariabeln första gången for-loopen körs.
slutvärde	Ett heltalsvärde som anger när loopen ska sluta köras, när loopvariabeln är större än slutvärdet kommer for-loopen INTE att utföras

```
for i := 0 to len(objekt.close)-1 do utvektor[i] := objekt.close[i]; end;
```

I detta exempel kommer for-loopen att köras lika många gånger som det finns platser i realvektorn objekt.close, och för varje plats i vektorn ge motsvarande värde till motsvarande plats i realvektorn utvektor

GetData s.60 - specialfall

```
Getdata( output, Databas, Vikingkod, Vektornamn, ””, ””, 0,0,0,0,0,0)
```

output	**I denna variabel sparas det data som hämtas ur databasen. Kan vara ett instrument, en realvector eller integer**
Databas	**Specifiera vilken databas datat ska hämtas ifrån: ”PriceData” för kursdatabasen, ”FundData” för fundamentaldatabasen och ”FundInfo” för speciella data som används i Fundamentalanalys**
Vikingkod	**Vikingkoden för det objekt som ska hämtas ur databasen.**
Vektornamn	**Namnet på den aktuella medlemsvektorn som ska hämtas för instrumentet (”close”, ”high”, ”low”, ”close”, ”open”, ”vol”, ”evol”) eller kortnamnet på en fundamentalvektor**
Getdata(rv, ”Pricedata”, ”190072”, ”close”, ””, ””, 0,0,0,0,0,0,0);	**Hämtar close-kurserna för objektet OMX Stockholm 30 och sparar datat i realvectorn rv**
Getdata(rv, ”Pricedata”, lista[i], ”high”, ””, ””, 0,0,0,0,0,0,0);	**Hämtar high-kurserna för objektet på plats i i objektlistan lista och sparar datat i realvectorn rv**

Komplett funktionsbeskrivning, appendix?

Getdata(output, Databas, Vikingkod, Vektornamn, String1, String2, FundIndex, Rapporttyp, Fundtidsenhet, Adjust, Integer1, Integer2, Felvariabel)

output	**I denna variabel sparas det data som hämtas ur databasen. Kan vara ett instrument, en realvector eller integer**
Databas	**Specifiera vilken databas datat ska hämtas ifrån: ”PriceData” för kursdatabasen, ”FundData” för fundamentaldatabasen och ”FundInfo” för speciella data som används i Fundamentalanalys**
Vikingkod	**Vikingkoden för det objekt som ska hämtas ur databasen.**

Vektornamn	**Namnet på den aktuella medlemsvektorn som ska hämtas för instrumentet (”close”, ”high”, ”low”, ”close”, ”open”, ”vol”, ”evol”) eller kortnamnet på en fundamentalvektor**
String1	**oanvänd strängparameter**
String2	**oanvänd strängparameter**
FundIndex	**Index för en fundamentalvektor**
Rapporttyp	**Typ av rapport vid laddning av fundamentaldata: 1 - Årsrapport, 2 - Kommunike och 3 - Prognos**
Fundtidsenhet	**Tidsperiod vid laddning av fundamentaldata: 1 - Månad, 2 - Kvartal och 3 - År**
Adjust	**Adjustfaktor vid laddning av fundamentaldata: 1 – Adjustfaktorn laddas, 2 – Fundamentaldatat laddas**
Integer1	**oanvänd heltalsparameter**
Integer2	**oanvänd heltalsparameter**
Felvariabel	**Valfri parameter, om den finns så ges den ett värde motsvarande felkoden nedan i händelse av ett fel vid körningen:**

Returvärden:

1. Inget fel
2. Typfel på Resultat-variabeln.
3. Databasen finns inte.
4. Ett objekt med given vikingkod finns inte.
5. En vektor med givet namn finns inte.
6. Dataladdning misslyckades. Ospecificerat fel.
7. GetData-laddning inte tillgänglig från denna applikation.
8. Aktuell tidsenhet (inte samma som fundtidsenhet) stöds inte vid denna typ av dataladdning. Exempelvis stöds inte laddning av fundamentaldata när man kör på intradaydata.
9. Varken Vektornamn eller FundIndex är givna vid fundamentalladdning.
10. Både Vektornamn och FundIndex är givna vid fundamentalladdning. Endast en i taget får anges.
11. Felaktig RapportTyp given. Den måste vara 1, 2 eller 3.
12. Felaktig Fundtidsenhet given. Den måste vara 1, 2 eller 3.
13. Felaktig Adjust given. Den måste vara 1 eller 2.
14. Givet Vektornamn saknas för Vikingkod vid fundamentalladdning.
15. Givet Fundindex saknas för Vikingkod vid fundamentalladdning.

If s.67

Funktionen len används för att ta reda på längden av en vektor eller en objektlista.

```
Len(vektor)
```

vektor	Vektor är en variabel av typen realvector, integervector, booleanvector eller objektlista
Len(main.close);	Returnerar ett heltalsvärde motsvarande längden på realvectorn main.close

Loop-satsen används för att upprepa en kodsekvens. En loop-sats innehåller inget villkor som while- och repeat-satserna. Repetitionen måste avbrytas med ett anrop till Exit-satsen. Loop-satsen måste avslutas med end;

```
loop // kod som skall upprepas exit; // Avbryt repetitionen end; i := 50; loop rvLoop[i] := 25.3; i := i + 2; if i >= 100 then exit; // Avbryt
```

RemoveFromSet lägger till vikingkodsträngar från en objektlista.

```
AddToSet(set, index);
```

set	En variable av datatyp set, d.v.s en objektlista
index	En variabel av datatyp integer, som beskriver positionen i objektlistan.
RemoveFromSet(list1, 78);	Tar bort det 78:e objektet från objektlistan list1

Repeat-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta till dess att villkoret är sant. Den andra skillnaden mot while-satsen är att villkoret för repetitionen ligger efter kodsekvensen. Detta innebär att satserna alltid kommer att utföras minst en gång. Repeat-satsen måste avslutas med end;.

```
repeat // kod som skall upprepas until krav för avslut ; i := 50; repeat rvRepeat[i] := 25.3; i := i + 2; until i >= 100;
```

Vartannat element i rvLoop från och med 50 till och med 98 sätts till 25.3. Samma som i loop-satsen ovan.

Return-satsen avslutar körningen av en funktion. Den finns i två varianter, en med parameter och en utan.

```
par() : datatyp som ska returneras; begin // Kod som utförs return variabel av rätt datatyp; // Kod efter return kommer ej utföras end; par(ob
```

Denna kod returnerar objektets volym som är en vektor av typen realvector. På detta sätt används return om man vill använda koden som n funktion som kan anropas av andra funktioner.

```
par(objekt : instrument; out volym : realvector); begin volym := modeller.objektvolym(objekt); end;
```

Om funktionen ovan kallas objektvolym() och ligger i katalogen modeller, så kan man med koden till vänster ge vektorn volym värden från objektets volymvektor indirekt genom att anropa funktionen objektvolym().

Timevec är en vektor vars element är av typen Time. Den innehåller tidpunkterna för motsvarande index i övriga vektorer. I den nuvarande versionen av modellspråket kan

dock inte Timevec[n] användas för att hitta tidpunkten för det n:te elementet. Istället måste TimeOfIndex-funktionen användas. Timevec används som parameter i några funktioner, exempelvis IndexOfTime, TimeOfIndex och TimeUnit.

```
TimeUnit(timevec);
```

Funktionen TimeUnit() returnerar en integer med timevec som inparameter. Värdet på integer beror på vilken period som timevec har, enligt:

1. Intradaydata
2. Dagsdata
3. Veckodata
4. Månadsdata

Funktionen Undef används för att undersöka om en variable är odefinierad. Resultatet av funktionen är true om variabeln som angivits som parameter till funktionen är odefinierad.

```
undef(variabel); r := 0/0; if undef(r) then // Detta är sant eftersom division // med noll ger odefinierat resultat. end;
```

Den felaktiga divisionen med 0 kommer att ge variabeln r ett odefinierat värde. Resultatet av undef(r) är således true och koden efter then kommer att utföras.

While-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta så länge villkoret är sant. Om villkoret för repetition är falskt första gången kommer kodsekvensen mellan while och end aldrig att köras. While-satsen måste avslutas med end;.

```
while villkorssats do // kod som ska utföras end; i := 50; while i < 100 do rvWhile[i] := 25.3; i := i + 2; end;
```

artannat element i rvLoop från och med 50 till och med 98 sätts till 25.3. Samma som i repeat-satsen ovan.

Returvärden:

1. Inget fel
2. Typfel på Resultat-variabeln.
3. Databasen finns inte.
4. Ett objekt med given vikingkod finns inte.
5. En vektor med givet namn finns inte.
6. Dataladdning misslyckades. Ospecificerat fel.
7. GetData-laddning inte tillgänglig från denna applikation.
8. Aktuell tidsenhet (inte samma som fundtidsenhet) stöds inte vid denna typ av dataladdning. Exempelvis stöds inte laddning av fundamentaldata när man kör på intradaydata.
9. Varken Vektornamn eller FundIndex är givna vid fundamentalladdning.
10. Både Vektornamn och FundIndex är givna vid fundamentalladdning. Endast en i taget får anges.
11. Felaktig RapportTyp given. Den måste vara 1, 2 eller 3.
12. Felaktig Fundtidsenhet given. Den måste vara 1, 2 eller 3.
13. Felaktig Adjust given. Den måste vara 1 eller 2.
14. Givet Vektornamn saknas för Vikingkod vid fundamentalladdning.
15. Givet Fundindex saknas för Vikingkod vid fundamentalladdning.

If s.67

Funktionen len används för att ta reda på längden av en vektor eller en objektlista.

```
Len(vektor)
```

vektor	Vektor är en variabel av typen realvector, integervector, booleanvector eller objektlista
Len(main.close);	Returnerar ett heltalsvärde motsvarande längden på realvectorn main.close

Loop-satsen används för att upprepa en kodsekvens. En loop-sats innehåller inget villkor som while- och repeat-satserna. Repetitionen måste avbrytas med ett anrop till Exit-satsen. Loop-satsen måste avslutas med end;

```
loop // kod som skall upprepas exit; // Avbryt repetitionen end; i := 50; loop rvLoop[i] := 25.3; i := i + 2; if i >= 100 then exit; // Avbryt
```

RemoveFromSet lägger till vikingkodsträngar från en objektlista.

```
AddToSet(set, index);
```

set	En variable av datatyp set, d.v.s en objektlista
index	En variabel av datatyp integer, som beskriver positionen i objektlistan.
RemoveFromSet(list1, 78);	Tar bort det 78:e objektet från objektlistan list1

Repeat-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta till dess att villkoret är sant. Den andra skillnaden mot while-satsen är att villkoret för repetitionen ligger efter kodsekvensen. Detta innebär att satserna alltid kommer att utföras minst en gång. Repeat-satsen måste avslutas med end;.

```
repeat // kod som skall upprepas until krav för avslut ; i := 50; repeat rvRepeat[i] := 25.3; i := i + 2; until i >= 100;
```

Vartannat element i rvLoop från och med 50 till och med 98 sätts till 25.3. Samma som i loop-satsen ovan.

Return-satsen avslutar körningen av en funktion. Den finns i två varianter, en med parameter och en utan.

```
par() : datatyp som ska returneras; begin // Kod som utförs return variabel av rätt datatyp; // Kod efter return kommer ej utföras end; par(ob
```

Denna kod returnerar objektets volym som är en vektor av typen realvector. På detta sätt används return om man vill använda koden som n funktion som kan anropas av andra funktioner.

```
par(objekt : instrument; out volym : realvector); begin volym := modeller.objektvolym(objekt); end;
```

Om funktionen ovan kallas objektvolym() och ligger i katalogen modeller, så kan man med koden till vänster ge vektorn volym värden från objektets volymvektor indirekt genom att anropa funktionen objektvolym().

Timevec är en vektor vars element är av typen Time. Den innehåller tidpunkterna för motsvarande index i övriga vektorer. I den nuvarande versionen av modellspråket kan dock inte Timevec[n] användas för att hitta tidpunkten för det n:te elementet. Istället måste TimeOfIndex-funktionen användas. Timevec används som parameter i några funktioner, exempelvis IndexOfTime, TimeOfIndex och TimeUnit.

```
TimeUnit(timevec);
```

Funktionen TimeUnit() returnerar en integer med timevec som inparameter. Värdet på integer beror på vilken period som timevec har, enligt:

1. Intradaydata
2. Dagsdata
3. Veckodata
4. Månadsdata

Funktionen Undef används för att undersöka om en variable är odefinierad. Resultatet av funktionen är true om variabeln som angivits som parameter till funktionen är odefinierad.

```
undef(variabel); r := 0/0; if undef(r) then // Detta är sant eftersom division // med noll ger odefinierat resultat. end;
```

Den felaktiga divisionen med 0 kommer att ge variabeln r ett odefinierat värde. Resultatet av undef(r) är således true och koden efter then kommer att utföras.

While-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta så länge villkoret är sant. Om villkoret för repetition är falskt första gången kommer kodsekvensen mellan while och end aldrig att köras. While-satsen måste avslutas med end;.

```
while villkorssats do // kod som ska utföras end; i := 50; while i < 100 do rvWhile[i] := 25.3; i := i + 2; end;
```

artannat element i rvLoop från och med 50 till och med 98 sätts till 25.3. Samma som i repeat-satsen ovan.

Konstanter

Minutes är en konstant som innehåller perioden för intradaydata. Minutes är av typen integer Konstanten now används för att hitta dagens värde i vektorer. Beroende på tidseneheten innehåller now index för aktuell tidpunkt, dag, vecka eller månad. Now är av typen integer.

Från en första titt

Egna funktioner som man vill använda på detta sätt skapar genom att definiera [returtyp] och använda return-kommandot i funktionen.

```
par(identifierare1 : datatyp1; identifierare2 : datatyp2;) : [returdatatyp];

var identifierare3 : datatyp3; begin .. [programkod] .. return identifierare3; end;
```

Exempel: En funktions som tar en en realvektor som inparameter och multiplicerar all data i vektorn med ett heltal

```
par(indata : realvector; term : integer) : realvector; var utvektor : realvector; begin utvektor := indata*term; return utvektor; end;
```

Om man sparer ner funktionen ovan med namnet ‘exempel’, under katalogen ‘Modeller’ så kan man senare anropa funktionen med modeller.exempel([realvector], [integer]), som nedan:

```
par(main : instrument; out utdata : realvector); begin utdata := modeller.exempel(main.close,2); end;
```

Där denna funktion använder sig av funktionen exempel för att multiplicera det aktuella objektets slutkurs med 2.

5. Operatorer

a. Precedens

När du skriver en funktion, som innehåller operatorer (not, *, +,> m.fl.) så måste du veta i vilken ordning som operationerna utförs. Beräkningsordningen i Futurelook är som följer:

Utförs först Fallande prioritetsordning (precedens) Utförs sist 1 2 3 4 Not * / and + - or = < > <= >=

Risken för fel undviker du genom flitigt bruk av paranteser. Operatorer med samma företräde (precedens) utförs i ett uttryck från vänster till höger.

T.ex. if-satsen

```
if x = 12 and b < 100
```

kommer inte att fungera eftersom ’and’ har högst precedens och det första som datorn ser är alltså ’12 and b’ , vilket är nonsens. Däremot if-satsen

```
if (x = 12) and (b < 100)
```

kommer att utföras korrekt.

Om minst en av operanderna är en vektor, så kommer operationerna att utföras element för element i vektorn (vektorerna). Antag t.ex. att vektorn 3, 4, 8 skall multipliceras med 4 så blir resultatet en ny vektor 12, 16, 32.

b. Aritmetiska operatorer

Dessa operatorer är matematiska operatorer som motsvarar de 4 räknesätten, nämligen plus (+), minus (-), multiplikation (*) och division (/). De variabeltyper som kan vara operander (som man tillämpar de aritmetiska operator på) är real och integer, samt vektorformen av dessa. Om båda operanderna är odefinierade blir resultatet

odefinierbart (undef). Likaså division med 0. Om operatorn minus (-) används för att beteckna ett negativt tal, s.k unärt minus, så måste parenteser användas.

2 + 3; 2 plus 3, resultatet av operatorn är 5
-2 * 3; Unärt minus är ok i början av en rad. Resultatet blir -6
3 * -2; Ej ok, måste separera unärt minus från andra operatorer
3 * (-2); Helt ok, unärt minus kan användas om man separerar den från andra operatorer med parenteser.
real / 0; Division med 0 är odefinierat!

Beroende på hur man kombinerar olika datatyper och operatorer så får man resultat av olika datatyper.

Denna kombination av operander	Ger denna datatyp för resultatet
integer * + - integer	integer
integer * + - real	real
integer * + - integervector	integervector
integer * + - realvector	realvector
real * + - integer	real
real * + - real	real
real * + - integervector	realvector
real * + - realvector	realvector
integervector * + - integer	integervector
integervector * + - real	realvector
integervector * + - integervector	integervector
integervector * + - realvector	realvector
realvector * + - integer	realvector
realvector * + - real	realvector
realvector * + - integervector	realvector
realvector * + - realvector	realvector
integer / integer	real
integer / real real	
integer / integervector	realvector
integer / realvector	realvector
real / integer	real
real / real	real
real / integervector	realvector
real / realvector	realvector
integervector / integer	realvector
integervector / real	realvector
integervector / integervector	realvector
integervector / realvector	realvector
realvector / integer	realvector
realvector / real	realvector
realvector / integervector	realvector
realvector / realvector	realvector

c. Booleanska operatorer

De boolska operatorer som Futurelook stöder är and och or. A and B; returnerar true endast om både A och B är true A or B; returnerar true om en av A eller B är true

Operanderna kan vara av typen boolean eller booleanvector. Ofta är operanderna satser som räknas ut och returnerar en av dessa två datatyper. Viktigt att tänka på är att båda operanderna alltid beräknas även om resultatet av beräkningen inte påverkas av den andra operandens värde. (enkelt uttryck) and (komplicerat uttryck); Operationen returnerar true endast om både det enkla och det komplicerade uttrycket är true. Detta innebär att programmet ofta räknar ut det komplicerade uttrycket även om det enkla redan returnerat false, helt i onödan. Nedan följer en sammanställning på resultatet av operatorerna and och or beroende på om A och B är true eller false.

A	Operator	B	Resultat
true	and	true	true
true	and	false	false
false	and	true	false
false	and	false	false
true	or	true	true
true	or	false	true
false	or	true	true
false	or	false	false

Beroende på hur man kombinerar olika datatyper och operatorer så får man resultat av olika datatyper.

boolean and or boolean	boolean
boolean and or booleanvector	booleanvector
booleanvector and or boolean	booleanvector
booleanvector and or booleanvector	booleanvector

d. Relationella operatorer

Dessa operatorer är jämförande opertorer som agerar på två operander. Om båda operanderna är odefinierade blir resultatet odefinierbart (undef). De operatorer som är tillgängliga i Futurelook är:

A = B; returnerar true om A har samma värde som B
A <> B; returnerar true om A ej har samma värde som B
A < B; returnerar true om A är mindre än eller lika med B
A > B; returnerar true om A är större än eller lika med B
A >= B; returnerar true om A är större än B
A <= B; returnerar true om A är mindre än B

Denna kombination av operander	Ger denna datatyp för resultatet
integer = <> integer	boolean
integer = <> real	boolean
integer = <> integervector	booleanvector
integer = <> realvector	booleanvector
real = <> integer	boolean
real = <> real	boolean
real = <> integervector	booleanvector
real = <> realvector	booleanvector
boolean = <> boolean	boolean
boolean = <> booleanvector	booleanvector
integervector = <> integer	booleanvector
integervector = <> real booleanvector	
integervector = <> integervector	booleanvector
integervector = <> realvector	booleanvector
realvector = <> integer	booleanvector
realvector = <> real	booleanvector
realvector = <> integervector	booleanvector
realvector = <> realvector	booleanvector
booleanvector = <> boolean	booleanvector
booleanvector = <> booleanvector	booleanvector
time = <> time	boolean
integer < <= > >= integer	boolean
integer < <= > >= real	boolean
integer < <= > >= integervector	booleanvector
integer < <= > >= realvector	booleanvector
real < <= > >= integer	boolean
real < <= > >= real	boolean
real < <= > >= integervector	booleanvector
real < <= > >= realvector	booleanvector
integervector < <= > >= integer	booleanvector
integervector < <= > >= real	booleanvector
integervector < <= > >= integervector	booleanvector
integervector < <= > >= realvector	booleanvector
realvector < <= > >= integer	booleanvector
realvector < <= > >= real	booleanvector
realvector < <= > >= integervector	booleanvector
realvector < <= > >= realvector	booleanvector
time < <= > >= time	boolean

6. Funktioner

1. a. Arbeta med funktioner och dess argument
2. b. Standardargument
3. c. Recursion
4. d. Felhantering

50 st

ABS, ACCUM, ALT, BMAX, BMAXD, BMIN, BMIND, BOT, BOTD, CONST, DiffFromSignal, EXP, FILL, FILTERBUY, FILTERSELL, LOG, MAVC, MAVN, MAVW, MAVX, MAX, MAXD, MIN, MIND, MOMABS, MOMREL, NORM, ONE, OSC, REL, RELV, REVERSE, ROUND, ROUNDS, SHIFT, SIGN, SMOOTH, SQRT, SQRTN, STDEV, STEP, STRINGCOMPARE, SUM, TIMEUNIT, TOP, TOPD, TRUNCATE, VMAVN, VSHIFT, XOR

Std.ABS

```
X := ABS(Y);
```

Returnerar absolutbeloppen av alla element i Y som element i vektorn X. ABS används exempelvis då man vill räkna med storleken på rörelser i en aktie, och inte på om rörelsen är positiv eller negativ.

bild ABS

```
par(main : instrument; out X : realvector); var begin X := ABS(main.close); end;
```

tabellresultat ABS

Std.ACCUM

```
Z := ACCUM(A, X, Y);
```

Returnerar ackumulerad summa av elementen i vektorerna X och Y. Om elementet A[i] är sant så adderas X[i] till summan och om A[i] är falskt så adderad Y[i] till summan. Odefinierade värden i X och Y räknas som 0.

bild ACCUM

```
par(main, main2 : instrument; out A : booleanvector; out Z : realvector); var X, Y : realvector; begin X := main.close; Y := main2.close; A :=
```

tabellresultat ACCUM

Std.ALT

```
Z := ALT(A, X, Y);
```

Returnerar element från X och Y till vektorn Z genom att för varje element låta vektorn A bestämma vilken. Om A[i] är true så kopieras värdet X[i] till Z[i] och om A[i] är falskt så kopieras Y[i] till Z[i]. ALT används till exempel om du vill rensa en vektor från värden som uppfyller vissa kriterier för att spara andra som uppfyller andra kriterier.

bild Std.ALT

```
par(main, main2 : instrument; out A : booleanvector; out Z : realvector); var X, Y : realvector; begin X := main.close; Y := main2.close; A :=
```

tabell Std.ALT

BMAX

```
Y := BMAX(X, A);
```

Returnerar högsta värdet på vektorn X räknat sedan vektorn A returnerat true. Används t.ex. för att hålla koll på high watermark eller drawdown för en rörelse sedan man fått köpsignal.

```
par(main : instrument; out Y : realvector; out BUY, SELL : booleanvector); var b3, b4 : booleanvector;  begin b3 := std.MAVN(main.close, 2) >
```

tabell BMX

BMAXD

```
Y := BMAXD(X, A);
```

Returnerar avståndet i perioder från senaste lokala top sedan A returnerat true. Används t.ex. för att räkna ut hur många dagar sedan rörelsen nådde sin högsta punkt efter att man fått en köpsignal.

```
par(main : instrument;  out Y : realvector; out BUY, SELL : booleanvector); var b3, b4 : booleanvector; begin b3 :=std.MAVN(main.close, 2) > s
```

tabell BMAXD

BMIN

```
Y := BMIN(X, A);
```

Som BMAX ovan.

```
par(main : instrument;  out Y : realvector; out BUY, SELL : booleanvector); var b3, b4 : booleanvector; begin b3 :=     std.MAVN(main.close, 2
```

tabell BMIN

BMIND

```
Y := BMIND(X, A);
```

Som BMIND ovan.

```
par(main : instrument;  out Y : realvector; out BUY, SELL : booleanvector); var b3, b4 : booleanvector; begin b3 :=     std.MAVN(main.close, 2
```

tabell BMIND

BOT

```
Y := BOT(X, p);
```

Returnerar värdet på det p:e lokala minimat i vektorn X innan X[i], där lokalt minima definieras som ett element vars värde är mindre än värden på elementet innan och efter. P- värde 1 anger att den senaste botten används, 2 att den näst senaste botten används osv.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.BOT(main.close,1);      end;
```

tabell

BOTD

```
Y := BOTD(X, p);
```

Returnerar avståndet i perioder från det p:e lokala minimum i vektorn X innan X[i], där lokalt minima definieras som ett element vars värde är mindre än värden på elementet innan och efter. P- värde 1 anger att den senaste botten används, 2 att den näst senaste botten används osv.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.BOTD(main.close,1); end;
```

tabell

CONST

```
Y := CONST(y);
```

Returnerar en realvektor där samtliga värden är konstanten y.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.CONST(-2.0); end;
```

DiffFromSignal

```
Y := DiffFromSignal(A, X);
```

Returnerar procentuella förändringen i värdena i vektorn X sedan A senast var true.

```
par(main : instrument;  out Y : realvector; out BUY, SELL : booleanvector); var b3, b4 : booleanvector; begin b3 :=     std.MAVN(main.close, 2
```

tabell

EXP

```
Y := EXP(X);
```

Returnerar värdet på exponentialfunktionen, ex med exponent x.

```
par(main : instrument; out Y : realvector); var begin Y := std.EXP(main.close); end;
```

tabell

FILL

```
Y := FILL(X);
```

Tilldelar Y[i] det senaste definierade värdet för vektorn X fram till X[i]. Funktionen används för att fylla ut luckor i en vektor X. Om det saknas data för en period så hämtar funktionen den senaste perioden med data och fyller igen luckan.

```
par(main : instrument; out Y : realvector); var begin Y := std.Fill(main.close); end;
```

FILTERBUY

```
C := FILTERBUY(A, B);
```

Returnerar en ny booleanvektor där endast nya true i vektorn A tas med sedan senaste true i vektorn B. Används vanligtvis för att filtrera fram en vektor C som endast ger true vid nya köpsignaler A sedan senaste säljsignalen i B, därav namnet FILTERBUY

```
par(main : instrument;  out Y : realvector; out b3,b4,BUY, SELL : booleanvector); var begin b3 :=       std.MAVN(main.close, 2) > std.MAVN(mai
```

FILTERSELL

```
C := FILTERSELL(A, B);
```

Returnerar en ny booleanvektor där endast nya true i vektorn B tas med sedan senaste true i vektorn A. Används vanligtvis för att filtrera fram en vektor C som endast ger true vid nya säljsignaler B sedan senaste köpsignalen i A, därav namnet FILTERSELL

```
par(main : instrument;  out Y : realvector; out b3,b4,BUY, SELL : booleanvector); var begin b3 :=       std.MAVN(main.close, 2) > std.MAVN(mai
```

(tabell)

LOG

```
Y := LOG(X);
```

Returnerar den naturliga logaritmen (basen e) för varje element i vektorn X. Notera att logaritmen av ett negativt tal är odefinierat.

```
par(main : instrument; out Y : realvector); var begin Y := std.LOG(main.close); end;
```

MAVC

```
Y := MAVC(X, p);
```

Returnerar ett centrerat medelvärde för vektorn X över perioden p. Det centrerade medelvärdet innebär att medelvärdet av vektorn X över perioden p kommer att placeras i Y vektorn på det element som motsvarar mitten av det mätområde som perioden p representerar.

```
par(main : instrument; out Y : realvector); var begin Y := std.MAVC(main.close,5); end;
```

MAVN

```
Y := MAVN(X, p);
```

Returnerar ett glidande medelvärde för vektorn X över perioden p. Medelvärdet av vektorn X över perioden p kommer att placeras i Y vektorn på det sista elementet som ingår i perioden p.

```
par(main : instrument; out Y : realvector); var begin Y := std.MAVN(main.close,5); end;
```

MAVW

```
Y := MAVW(X, p);
```

Returnerar det viktade medelvärdet av vektorn X över perioden p. Det viktade medelvärdet ger det sista elementet störst vikt och vikten avtar linjärt ner till det sista

elementet som ingår i beräkningen vars vikt är lägst.

```
par(main : instrument; out Y : realvector); var begin Y := std.MAVW(main.close,5); end;
```

MAVX

```
Y := MAVX(X, p)
```

Returnerar det exponentiellt vägda glidande medelvärdet för vektorn X över perioden p. Det exponentiella medelvärdet viktar de senaste värdena högre än tidigare värden och vikten avtar exponentiellt ju äldre datat blir.

par(main : instrument; out Y : realvector); var begin Y := std.MAVX(main.close,5); end;

MAX

```
Y := MAX(X, p);
```

Returnerar det högsta värdet i vektorn X sett över perioden p.

```
par(main : instrument; out Y : realvector); var begin Y := std.MAX(main.close,5); end;
```

MAXD

```
Y := MAXD(X, p);
```

Returnerar avståndet i antal perioder till det högsta värdet sett över perioden p.

```
par(main : instrument; out Y : realvector); var begin Y := std.MAXD(main.close,5); end;
```

MIN

```
Y := MIN(X, p);
```

Returnerar det lägsta värdet i vektorn X sett över perioden p.

```
par(main : instrument; out Y : realvector); var begin Y := std.MIN(main.close,5); end;
```

MIND

```
Y := MIND(X, p);
```

Returnerar avståndet i antal perioder till det lägsta värdet sett över perioden p.

```
par(main : instrument; out Y : realvector); var begin Y := std.MIND(main.close,5); end;
```

MOMABS

```
Y := MOMABS(X, p);
```

Returnerar momentum av X under perioden p, dvs. differensen mellan värdet på element i och värdet på element i – p. Notera att namnet innehåller ABS för att skilja på absolut och relativ rörelse, inte för att de returnerade värdena är absoluta. De returnerade värdena kan vara både positiva och negativa.

```
par(main : instrument; out Y : realvector); var begin Y := std.MOMABS(main.close,5); end;
```

MOMREL

```
MOMREL
```

Returnerar relativtmomentum av X under perioden p, dvs. differensen mellan värdet på element i och värdet på element i – p uttryckt i procent.

```
par(main : instrument; out Y : realvector); var begin Y := std.MOMREL(main.close,5); end;
```

NORM

```
Z := NORM(X, Y, a);
```

Om a sätts till 1 så returnerar funktionen en realvektor Z med startvärdet från vektorn X men den relativa utvecklingen från vektorn Y. Sätts a till 0 så kopieras Y till Z.

```
par(main, main2 : instrument; out Y : realvector); var begin Y := std.NORM(main.close, main2.close,1); end;
```

ONE

```
Y := ONE(A);
```

Funktionen returnerar en numerisk vector Y från den booleanska vektorn A. Alla element i A som är true ger värdet 1 och de som är false ger värdet 0 för motsvarande element i Y.

```
par(main : instrument;  out Y : realvector; out b : booleanvector); var begin   b := std.MAVN(main.close, 2) > std.MAVN(main.close, 4); Y := S
```

OSC

```
Y := OSC(X, l, s);
```

Returnerar differansen mellan det glidande medelvärdet av X med perioden l och det glidande medelvärdet av X med perioden s. Man kan se detta som en sammanslagning av två MAVN-funktioner enligt Y := MAVN(X,l) – MAVN(X,s);

```
par(main : instrument; out Y : realvector); var begin Y := std.OSC(main.close,2,6); end;
```

REL

```
Z := REL(X, Y);
```

Returnerar X utveckling relativt Y uttryckt numeriskt, alltså med start i 1.0

```
par(main, main2 : instrument; out Y : realvector); var begin Y := std.REL(main.close, main2.close); end;
```

RELV

```
Y := RELV(X);
```

Returnerar en normaliserad vektor med start i 1.0 och som utvecklas enligt X.

```
par(main : instrument; out Y : realvector); var begin Y := std.RELV(main.close); end;
```

REVERSE

```
Y := REVERSE(X);
```

Funktionen returnerar en vektor Y som är omvänd jämfört med vektorn X. Första elementet i Y motsvaras av sista elementet i X och så vidare tills det sista elementet i Y som motsvaras av det första elementet i X.

```
par(main : instrument; out Y : realvector); var begin Y := std.REVERSE(main.close); end;
```

ROUND

```
Y := ROUND(X);
```

Returnerar en integervektor Y som innehåller värdena I X som elemtent för element har avrundats till närmaste heltal. Decimaltalet 1.5 avrundas uppåt till 2.0. ROUND tar en vektor som inparameter och returnerar en vektor.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.ROUND(main.close); end;
```

ROUNDS

```
i := ROUNDS(r);
```

Returnerar integer som motsvarar avrundning av real-variabeln r. ROUNDS tar ett realtal som inparameter och returnerar en integer.

```
par(main : instrument;  out Y : integervector); var i : integer;        begin for i := 0 to len(main.close)-1 do Y[i] := Std.ROUNDS(main.close
```

SHIFT

```
Y := SHIFT(X, p);
```

Returnerar en vector Y bestående av värden i X som flyttats p perioder fram i tiden. Element från X som skulle hamna utanför vektorn Y faller bort.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.SHIFT(main.close,5); end;
```

SIGN

```
Y := SIGN(X);
```

Returnerar en vektor Y som innehåller -1, 0 och 1 beroende på huruvida motsvarande element i vektorn X är negativt, 0 eller positivt.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.SIGN(main.close); end;
```

SMOOTH

```
Y := SMOOTH(X, f);
```

Returnerar en vector Y som i en stigande trend för X ges av högsta värdet i X sedan trenden inletts. Den stigande trenden bryts då en rörelse sker mot trenden med mer än f %. Eftersom trenden då övergår i en fallande trend kommer Y motsvara de lägsta värdena i X sedan den trenden påbörjades och kommer så vara tills X har en rörelse mot den fallande trenden med f % eller mer.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.SMOOTH(main.close,2.0); end;
```

SQRT

```
Y := SQRT(X);
```

Returnerar en vektor Y med element bestående av kvadratroten av motsvarande element i X. Notera att kvadratroten av negativa tal är odefinierade.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.SQRT(main.close); end;
```

SQRTN

```
Y := SQRTN(X, k);
```

Returnerar en vektor Y med element bestående av den n:te roten av motsvarande element i X.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.SQRTN(main.close,3); end;
```

STDEV

```
Y := STDEV(X, p);
```

Returnerar standardavvikelsen för vektorn X över perioden p.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.STDEV(main.close,5); end;
```

STEP

```
Y := STEP(A,X);
```

Returnerar en vektor Y som element för element tilldelas värden ur X om A är sann och föregående värde från Y om A är falsk.

```
par(main : instrument; out A : booleanvector;   out Y : realvector); var begin  A := std.MAVN(main.close, 2) > std.MAVN(main.close, 4); Y := S
```

STRINGCOMPARE

```
x := STRINGCOMPARE(str1, str2);
```

Returnerar en integer x som är -1, 0 eller 1 beroende på om str1 kommer före, är lika med eller kommer efter str2, om man skulle sortera dem i bokstavsordning. ”Automobil” kommer alltså före ”Bilbana” men efter ”Arbete”.

```
par(main : instrument; out test1, test2, test3 : integervector); var str1, str2, str3 : string; begin str1 := "Automobil"; str2 := "Bilbana";
```

SUM

Returnerar summan av alla element i X under de senaste p perioderna

```
par(main : instrument; out X : realvector); var begin X := SUM(main.close,5); end;
```

TIMEUNIT

```
i := TIMEUNIT(timevec);
```

Returnerar ett heltal beroende av den period som timevec använder. Är datat intradaydata får i värdet 1. Är perioden dags-, vecko- eller månadsdata får i värdet 2,3 eller 4.

```
par(main : instrument; period, MAVPeriod1, MAVPeriod2 : integer; out volatility, mvvol1, mvvol2 : realvector); var timefactor : integer; begin
```

TOP

```
Y := TOP(X,p);
```

Returnerar det p:te lokala maximum i vektorn X. Ett lokalt maximum definieras som ett element vars värde är större än värden på elementet innan och efter. p = 1 anger att den senaste toppen används, 2 att den näst senaste toppen används osv.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.TOP(main.close,1); end;
```

TOPD

```
Y := TOPD(X, p);
```

Returnerar avståndet i perioder från det p:e lokala maximum i vektorn X innan X[i], där lokalt maxima definieras som ett element vars värde är större än värden på elementet innan och efter. P- värde 1 anger att den senaste toppen används, 2 att den näst senaste toppen används osv.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.TOPD(main.close,1); end;
```

TRUNCATE

```
K := TRUNCATE(X);
```

Returnerar en integervektor där varje element är trunkerade från elementen i realvektorn X. Detta innebär att alla decimaler har kapats och endast talens heltalskomponent behålls.

```
par(main : instrument;  out Y : realvector); var begin Y := Std.TRUNCATE(main.close); end;
```

VMAVN

```
Z := VMAVN(X, Y);
```

Returnerar ett aritmetiskt medelvärde av X, jmfr. MAVN(X, p) , men där perioden p kan variera. För varje element i Z[i] räknas istället medelvärdet av X[i] med perioden Y[i] ut.

```
par(main : instrument; out Z : realvector; out X : realvector); var i : integer; begin X := 5; for i:= 0 to len(main.close)-10 do X[i] := 3; e
```

VSHIFT

```
Z := VSHIFT(X,Y);
```

Returnerar en vektor Z som innehåller elementen från X flyttade Y antal perioder fram i tiden. Denna funktion liknar SHIFT(X, p) , men antalet steg som elementen i X ska flyttas framåt kan ändra över tiden och definieras i Y.

```
par(main : instrument; out Z : realvector; out X : realvector); var i : integer; begin X := 5; for i:= 0 to len(main.close)-10 do X[i] := 3; e
```

XOR

```
C := XOR(A, B);
```

Returnerar den logiska operatorn exklusiv OR kört på de två booleanska vektorerna A och B. Element för element kommer XOR funktionen att returnera true om antingen A eller B är true och endast returnera false om både A och B returnerar false.

```
par(main : instrument;  out C, b3, b4 : booleanvector); var begin b3 := std.MAVN(main.close, 2) > std.MAVN(main.close, 4); b4 := std.MAVN(main
```

7. Programmering i praktiken

I praktiken kommer programmen som du gör att fokusera på hämtningen och bearbetningen av en eller flera vektorer bestående av finansiella data. En överblicksbild av en modell (eller projektplanen för en modell som ska programmeras) kunde se ut så här:

- Hämtning av data
- Bearbetning av data
- Presentation av data

Hämtning av data: Hur skall modellen komma åt den data som den behöver för att arbeta. De två viktigaste sätten att hämta data är:

- Använda sig av objekt som definieras i par() segmentet och som t.ex. kan bestämmas från gång till gång då modellen körs genom att välja ’aktuellt objekt’ eller specificera en mängd objekt i modellinställningarna.
- Använda sig av Vikingens interna databas, på så sätt att modellen hämtar data med funktionen Getdata() och ger dess värden till en vektor definierad i modellen

```
par(main1 : instrument; out instr1, instr2 : realvector); var begin instr1 := main1.close; GetData( instr2, "PriceData", "190072", "close","",
```

I exemplet ovan definieras main1 som ett instrument I par() segmentet. Detta ser säkert bekant ut och är det vanliga sättet att bygga en modell, man vill ju att vikingens ’aktuellt objekt’ ska vara det som bestämmer hur vi hämtar data till modellen. Vi ser hur man i koden gör för att arbeta med denna data. Genom att använda

```
instr1 := main1.close;
```

kan vi på enklast möjliga sätt komma åt datavektorn med close-kurser. Modellen hämtar close-kursen från objektet main1 och ger denna data till vektorn instr1, så att modellen kan bearbeta datat. Det andra sättet att hämta data ser vi längre ner i modellen. Funktionen GetData() används här för att komma åt datavektorn som motsvarar close-kurserna för objektet med vikingkoden ”190072” dvs. OMX Stockholm 30. Close-kursen för OMX Stockholm 30 ges till vektorn instr1 för att bearbetas vidare i modellen.

Bearbetning av data: Huvuddelen av arbetet som du kommer att lägga ner på att programmera modeller i Futurelook kommer sannolikt att handla om bearbetning av datat inom ramen för modellen. Det är inte möjligt att redogöra för allt som kan åstadkommas med programmeringsspråket, men vi kan i detta kapitel försöka lyfta fram de vanligaste problemen och hur de brukar lösas med de verktyg som finns till förfogande. Funktionsbiblioteket som beskrivs i avsnitt 6 ger användaren en stor verktygslåda för bearbetning av data, och dessa fördefinierade funktioner gör att en stor del av jobbet är gjort eftersom byggstenarna i en mer avancerad modell (medelvärden, min/max, moment) inte behöver programmeras varje gång. Det som kan ställa till problem för en nybörjare är dock de situationer där man behöver hantera enskilda element i vektorerna, något som man oftast gör genom att programmera en loop. I futurelook kan man använda sig av flera olika loopar: for-, loop-, repeat- och while-loopen. Att veta vilken man ska använda i vilken situation är inte helt lätt, men det är också så att det i flertalet fall fungerar lika bra med flera än en av dem (dock kanske inte lika effektivt.) Iden är ofta den att man vill titta på alla element i en vektor ett och ett, genom att loopa genom vektorn. Varje gång som loopen körs så kommer en loop-variabel att ha ett heltalsvärde som ökar med ett då loopen är klar.

```
for i := 1 to len(closevector) -1 do if closevector[i] > closevector[i-1] then stigande[i] := true; end; end;
```

I exemplet ovan vill vi loopa genom vektorn closevector för att hitta de tillfällen då värdet på ett element är högre än elementet innan. Då vi hittar en sådan situation vill vi att den markeras dom true på motsvarande position i en boolean-vektor. Första gången loopen körs kommer i att ha värdet 1, vilket gör att vi jämför closevector[1] med closevector[0] och ifall if-satsen utvärderas som true så kommer stigande[1] att ges värdet true. Då loopen körts en gång kommer i att få värdet 2 och all kod kommer att köras än en gång. På så sätt kommer alla element i vektorn att undersökas ett i gången. Presentation av data: se kapitel 8.