UNIVERSITEIT ANTWERPEN
FACULTEIT WETENSCHAPPEN
DEPARTEMENT WISKUNDE-INFORMATICA

OBERON CODE CONVENTIONS

Laatste aanpassing: 15 oktober 2003

Inhoudsopgave

1 Bestandsnamen

Geef Oberon-bestanden dezelfde naam als de module die je erin gedefinieerd hebt, gevolgd door het suffix `.Mod`. In elk *project* komt één `.Tool`-bestand voor dat de naam van het project draagt en volgende vier delen bevat:

- de commandos om te compileren
- het commando om te debuggen na een system trap
- het commando om *oude* versies uit het geheugen te wissen
- commando(s) om het programma op te starten met eventuele testwaarden

Zie sectie 9 (p. 8) achteraan voor een voorbeeld van een `.Tool`-bestand.

2 Organizatie Bestanden

Elk `.Mod`-bestand bevat volgende onderdelen:

(1) de *`MODULE`-syntax*

(2) het *algemene commentaarblok* waarin een korte beschrijving van de module wordt gegeven alsook de auteur en de datum van creatie

(3) eventueel een *`IMPORT`-lijst* (een alias voor een module-naam wordt niet toegelaten)

(4) eventueel een *declaratielijst* met in volgorde

   (a) constanten
   (b) types
   (c) variabelen
   (d) voorwaartse declaraties
   (e) proceduredeclaraties

(5) het *body*-gedeelte van de module

Plaats een witregel tussen elk van deze onderdelen. Zie sectie 9 (p. 8) voor een volledig uitgewerkt voorbeeld.

3 Indentatie

Enkel de onderdelen (1), (2) en (5) vermeld in vorige sectie worden volledig tegen de linker kantlijn geplaatst. Alle andere worden met één tabpositie geïndenteerd.

Op dezelfde manier worden de variabeledeclaraties van een procedure met een extra tabpositie geïndenteerd.

3.1 Regels Afbreken

Zorg ervoor dat de regels in uw code niet te lang worden (zodanig dat je niet moet gaan *scrollen* om een regel te lezen). Dikwijls is dit echter onvermijdelijk. Breek in die situaties de regel af en indenteer hem één tabpositie méér dan de eerste regel. Neem bij het afbreken volgende regels in acht:

- breek af *na* een komma
- breek af *voor* een operator
- verkies *higher-level*-afbreken boven *lower-level* (zoals bijvoorbeeld in lange expressies)

4 Commentaar

Op sommige plaatsen (zoals bij lange, ingewikkelde expressies) kan het nodig zijn een woordje commentaar toe te voegen. Aarzel niet dit dan ook te doen.

Procedures worden ook voorzien van een commentaarblok met volgende gegevens:

- *wat* deze procedure doet
- de verschillende *parameters* en wat ze voorstellen
- het *returntype* en wat het voorstelt
- het *algoritme* dat je gebruikt hebt (bijvoorbeeld Bubblesort)

5 Declaraties

5.1 Velden

Voor het aanroepen van velden van een record gebruikt men altijd de punt (`.`). Echter, om het onderscheid goed duidelijk te maken met het aanroepen van een veld via een pointer gebruiken we niet de standaard manier met de punt zoals (als `p` een pointer is)

```
p.next;
```

maar wel

```
pˆ.next;
```

6 Statements

Zet op elke regel slechts één *statement*. Dus niet

```
OutExt.String("Hello world!"); OutExt.Ln;
```

wel

```
OutExt.String("Hello world!");
OutExt.Ln;
```

6.1 Keuzeinstructies

6.1.1 Het `IF-THEN`-Statement

Het `IF-THEN`-statement heeft de volgende vorm:

```
IF (<conditie>) THEN
  <statements>
END;
```

6.1.2 Het `IF-THEN-ELSE`-Statement

Het `IF-THEN-ELSE`-statement heeft de volgende vorm:

```
IF (<conditie>) THEN
  <statements>
ELSE
  <statements>
END;
```

6.1.3 Het `IF-THEN-ELSIF-THEN`-Statement

Het `IF-THEN-ELSIF-THEN`-statement heeft de volgende vorm:

```
IF (<conditie>) THEN
  <statements>
ELSIF (<conditie>) THEN
  <statements>
END;
```

6.1.4 Het `CASE`-Statement

Het `CASE`-statement heeft de volgende vorm:

```
CASE <variabele> OF
  <1a>, <1b>, ...:
    <statements>
  | <2a>, <2b>, ...:
    <statements>
  ...
ELSE
  <statements>
END;
```

6.2 Lusinstructies

6.2.1 De `FOR`-Lus

De `FOR`-lus heeft de volgende vorm:

```
FOR <lusvariabele> := <beginwaarde> TO <eindwaarde> DO
  <statements>
END;
```

Voeg, indien nodig, het deel `BY <stap>` toe tussen `<eindwaarde>` en `DO`.

6.2.2 De `WHILE`-Lus

De `WHILE`-lus heeft de volgende vorm:

```
WHILE (<conditie>) DO
  <statements>
END;
```

6.2.3 De `REPEAT`-Lus

De `REPEAT`-lus heeft de volgende vorm:

```
REPEAT
  <statements>
UNTIL (<conditie>);
```

6.2.4 De `LOOP`-Lus

De `LOOP`-lus heeft de volgende vorm:

```
LOOP
  <statements>
  IF (<conditie>) THEN
    EXIT;
  END;
  <statements>
END;
```

Algemene opmerking bij de controleinstructies: zet een conditie steeds tussen ronde haakjes en indenteer de controlewoorden (zoals `IF`, `ELSE`, `ELSIF`, `END`, `WHILE`, ...) steeds even ver. Plaats ook (zelfs als het niet *moet*) een `;` na elke regel, behalve als het niet *hoort* zoals na `THEN`, `DO`, enz.

7 Spaties

Plaats een spatie op de volgende plaatsen:

- rond elke operator
- na elke komma
- in expressies
  - voor elke `(`
  - na elke `)`

8 Naamgevingen

Goede naamgevingen zijn zeer belangrijk en dus is het aangewezen zo weinig mogelijk afkortingen te gebruiken en namen zo logisch mogelijk te kiezen (dus zorg ervoor dat ze overeenstemmen met de realiteit).

Begin de naam van een module, procedure en nieuw type steeds met een hoofdletter. In geval van meerdere woorden worden deze onderscheiden door een hoofdletter. Dus niet

```
PROCEDURE absolutewaarde(x: INTEGER) : BOOLEAN;
```

wel

```
PROCEDURE AbsoluteWaarde(x: INTEGER) : BOOLEAN;
```

Variabelen beginnen daarentegen met een kleine letter. In geval van meerdere woorden geldt dezelfde regel als hierboven.

Gebruik voor constanten steeds allemaal hoofdletters. Hier worden echter verschillende woorden niet expliciet onderscheiden.

9 Voorbeeld

- Naam project: `OCC`
- Bestanden in project:
  - `OCC.Tool`

```
(*
  Author: Frederic Hancke
  Date Created: 30/09/2001
*)

Builder.Compile \s
  OutExt.Mod
  OCC.Mod
~


Builder.Compile \f *

System.Free
  OCC.Mod*
~


OCC.Run
  1
  "Hello World!"
~


OCC.Run
  2
  "Hello World!"
~
```

  - `OutExt.Mod` Niet zelf geschreven, dus wordt hier niet weergegeven.

  - `OCC.Mod`

```
MODULE OCC;

(*
  Description: Example module for OCC.
  Author: Frederic Hancke
  Date Created: 30/09/2001
*)

  IMPORT
    In, OutExt;

  CONST
    STRINGLENGTH = 80;

  TYPE
    String = ARRAY STRINGLENGTH OF CHAR;

  VAR
    myString: String;

  PROCEDURE ^ PrintMessage(VAR msg: String);

  PROCEDURE Run*;
  (*
    Description: This is the main procedure.
    Parameters: -
    Returntype: -
    Algorithm: -
  *)

    VAR
      i: INTEGER;
      s: String;

  BEGIN
    In.Open;
    In.Int(i);
    IF (i = 1) THEN
      In.String(s);
      PrintMessage(s);
    ELSE
      s := "";
      PrintMessage(s);
    END;
```

```
  END Run;

  PROCEDURE PrintMessage(VAR msg: String);
  (*
    Description: Prints a certain message.
    Parameters:
      msg: The message to print.
    Returntype: -
    Algorithm: -
  *)

  BEGIN
    OutExt.String(myString);
    OutExt.String(msg);
    OutExt.Ln;
  END PrintMessage;

BEGIN
  myString := "PrintMessage: ";
  OutExt.Open;
END OCC.
```