10. UNITS EN OVERLAYS

10.1 Inleiding

In de vorige hoofdstukken zijn we al een paar keer zijdelings het begrip "unit" tegengekomen. Bij het beheer van de schijf hadden we de unit DOS nodig en om met ClrScr het scherm schoon te kunnen vegen, en moest onder de naam van het programma de opdracht "USES CRT" opgenomen worden.

Wat zijn units nu eigenlijk en hoe worden ze gemaakt? Met units kun je een programmabibliotheek opbouwen, waarvan je de onderdelen in je programma kunt oproepen als je ze nodig hebt. Je kunt je voorstellen dat je in een programma regelmatig de tijd of de datum van vandaag in de vorm van een string nodig hebt. Het ligt dan voor de hand dat je functies gebruikt om deze strings te maken. Het is waarschijnlijk dat je deze functies ook in andere programma's nog eens nodig zult hebben. Om iedere keer het wiel uit te vinden en de functies opnieuw in een programma te zetten, is een beetje verspilling van tijd en werk. In Turbo Pascal is dat ook niet nodig. We kunnen deze functies onderbrengen in een unit. Als we dan de aangemaakte unit in de USES-regel vermelden, wordt de unit met ons programma meegecompileerd (zie ook paragraaf 10.3) en zijn de procedures en functies uit die unit in ons programma aan te roepen.

Een unit die je hebt gemaakt, moet je opslaan in de directory die je voor units hebt opgegeven onder «*Options*» en «*Directories*» (zie ook pagina 29).

Ook hier maakt een praktijkvoorbeeld het allemaal duidelijker:

UNIT UNIT_1;

```
Interface
USES DOS;

CONST
   UREN_IN_ETMAAL = 24;

FUNCTION DatumStr: String;
FUNCTION TijdStr : String;

Implementation

VAR
   SCHEIDINGTIJD, SCHEIDINGDATUM: Char;

FUNCTION NaamDag(D:Word): String;
BEGIN
   CASE D OF
      0: NaamDag := ’Zondag ’;
      1: NaamDag := ’Maandag ’;
      2: NaamDag := ’Dinsdag ’;
      3: NaamDag := ’Woensdag ’;
      4: NaamDag := ’Donderdag ’;
      5: NaamDag := ’Vrijdag ’;
      6: NaamDag :=’Zaterdag ’
   END
END;

FUNCTION DatumStr: String;
VAR
   DAG, MAAND,
   JAAR, DAG_VAN_DE_WEEK: Word;
   CONVERT              : Word;
   STR_CONVERT          : String[4];
   DATUM                : String[10];
   I                    : Byte;

   BEGIN
      DATUM := ’’;
      GetDate(JAAR,MAAND,DAG,DAG_VAN_DE_WEEK);
      FOR I := 1 TO 3 DO
      BEGIN
         CASE I OF
            1: CONVERT := DAG;
            2: CONVERT := MAAND;
            3: CONVERT := JAAR
```

```
         END;
         Str(CONVERT,STR_CONVERT);
         IF Length(STR_CONVERT)=1 THEN
         STR_CONVERT := '0' + STR_CONVERT;
         DATUM := DATUM + STR_CONVERT;
         IF I < 3 THEN DATUM := DATUM + SCHEIDINGDATUM
      END;
      DatumStr := NaamDag(DAG_VAN_DE_WEEK) + DATUM
   END;

FUNCTION TijdStr:String;
VAR
   UUR, MIN,
   SEC, HSEC  : Word;
   CONVERT    : Word;
   STR_CONVERT: String[4];
   TIJD       : String[8];
   I          : Byte;

BEGIN
   TIJD := '';
   GetTime(UUR,MIN,SEC,HSEC);
   FOR I:= 1 TO 3 DO
   BEGIN
      CASE I OF
         1: CONVERT := UUR;
         2: CONVERT := MIN;
         3: CONVERT := SEC
      END;
      Str(CONVERT,STR_CONVERT);
      IF Length(STR_CONVERT)=1 THEN
      STR_CONVERT := '0' + STR_CONVERT;
      TIJD := TIJD + STR_CONVERT;
      IF I < 3 THEN TIJD := TIJD + SCHEIDINGTIJD;
   END;
   TijdStr := TIJD
END;

BEGIN
   SCHEIDINGTIJD  := ':';
   SCHEIDINGDATUM := '-'
END.
```

Regels:Toelichting:

[1]1 Geef de unit de naam UNIT_1.
[2]2Begin van het interface-gedeelte.
[3]3 Gebruik de unit DOS.
[4]4-5Declareer een globale constante.
[5]6-7Declareer globale functies.
[6]8 Begin van het implementatie-gedeelte.
[7]9-10Declareer lokale variabelen van de unit.
[8]11-22Lokale functie NaamDag(D:Word): String;
[9]23-48 Function DatumStr: String;
[10] 49-74Function TijdStr: String;
[11]75-77 Initialiseer de variabelen van de unit.
[12] 78 Einde van de unit.

Toelichting:

[1]De naam die je in het programma aan de unit geeft en de naam waaronder je de unit in de unit-directory opslaat, moeten dezelfde zijn. Als deze namen niet gelijk zijn, wordt het compileren van een programma onderbroken door foutmelding 69: "unit name mismatch".

[2]Onmiddellijk na de naam begint het interface-gedeelte van de unit. Dit wordt aangegeven door het beschermde woord "interface". Procedures, functies en data die gedeclareerd zijn in het interface-gedeelte van de unit zijn globaal en kunnen dus door andere units en programma's gebruikt worden. Het interface-gedeelte wordt afgesloten met het beschermde woord "implementation".

[3]Direct achter de opdracht "Interface" staat de USES-regel. UNIT_1 maakt op zijn beurt gebruik van de unit DOS. De te gebruiken units worden hier achter elkaar gezet, gescheiden door een komma. De lijst wordt afgesloten met een puntkomma.

[4]De constante UREN_IN_ETMAAL is globaal gedeclareerd en kan daarom door andere units en programma's gebruikt worden.

[5]In het interface-gedeelte wordt slechts de kop van een procedure of functie opgenomen. De code van deze functies en procedures staat in het implementatie-gedeelte.

[6]Het begin van het implementatie-gedeelte wordt aangegeven door het beschermde woord "implementation", wat tevens de afsluiting is van het interface-gedeelte.

[7]Ook hier kunnen weer variabelen en constanten gedeclareerd worden. In tegenstelling tot constanten en variabelen die in het interface-gedeelte gedeclareerd zijn, zijn deze variabelen en constanten niet toegankelijk voor de programma's en/of units die deze unit gebruiken. De variabelen SCHEIDINGTIJD en

SCHEIDINGDATUM zijn voor alle procedures en functies binnen de unit toegankelijk. Procedures en functies van buiten de unit kunnen deze variabelen niet gebruiken.

[8]Dit geldt ook voor de functie NaamDag. Omdat NaamDag niet gedeclareerd is in het interface-gedeelte en dus alleen voorkomt in het implementatie-gedeelte, is NaamDag een lokale functie van UNIT_1. NaamDag krijgt als parameter een dagnummer door en retourneert een string die de naam van de dag bevat.

[9]De functie DatumStr is praktisch gelijk aan de functie DatumStr in het programma SCHIJF_2. Er is één uitzondering. In SCHIJF_2 wordt de datum als parameter doorgegeven. Hier wordt de systeemdatum opgevraagd. Hiervoor maken we gebruik van de in de unit DOS gedefinieerde procedure GetDate. GetDate krijgt VAR-parameters door voor JAAR, MAAND, DAG en DAG_VAN_DE_WEEK. Na terugkeer uit GetDate, bevatten deze lokale variabelen de systeemdatum. Als globale functies of procedures een parameterlijst hebben, moet deze in ieder geval vermeld worden in het interface-gedeelte. Bij het uitwerken van de code in het implementatie-gedeelte, mag de parameterlijst achterwege blijven, maar dat hoeft niet.

[10] De functie TijdStr roept in de unit DOS de procedure GetTime aan. Deze procedure werkt op dezelfde wijze als GetDate. Alleen retourneert GetTime de tijd in parameters voor het uur, de minuten, de seconden en honderdsten van seconden.

[11] Het laatste deel van de unit is het initialiserings-gedeelte. Hierin kunnen variabelen van de unit een beginwaarde krijgen, of kan een procedure of functie aangeroepen worden. In dit geval krijgen de lokale variabelen SCHEIDINGTIJD en SCHEIDINGDATUM een beginwaarde.

[12]Een unit eindigt met "END." Als er niets te initialiseren valt, wordt de unit afgesloten met alleen het woord "END." en kan het woord "BEGIN" achterwege blijven.

Het volgende programma heeft in zijn USES-lijst de unit UNIT_1 opgenomen en beschikt daarmee over de globale functies van UNIT_1 en over de daar gedeclareerde constanten. De lokale functies en variabelen van UNIT_1 kunnen door PROG_1 niet gebruikt worden:

```
PROGRAM PROG_1;
USES CRT, UNIT_1;

BEGIN
   ClrScr;
   Writeln('De tijd is      :',TijdStr);
   Writeln('De datum is     :', DatumStr);
   Writeln('Een etmaal telt :',UREN_IN_ETMAAL,' uur.');
```

```
   Readln
END.
```

10.2 Samenvatting unit-regels

Voor de duidelijkheid staan hier nog even de regels op een rijtje waar je je aan moet houden bij het werken met units:

- De unit moet worden opgeslagen in de unit-directory (zie ook pagina 29).
- De naam die je in het programma aan de unit geeft en de naam waaronder je de unit in de unit-directory opslaat, moeten dezelfde zijn.
- Na de naam van de unit wordt het beschermde woord "interface" opgenomen.
- In het interface-gedeelte worden globale constanten, variabelen, procedures en functies gedeclareerd.
- Van de globale functies en procedures wordt in het interface-gedeelte alleen de kop vermeld.
- Het beschermde woord "implementation" geeft aan waar het implementatie-gedeelte begint.
- In het implementatie-gedeelte kunnen constanten, variabelen, procedures en functies lokaal gedeclareerd worden. De procedures en functies waarvan de kop in het implementatie-gedeelte vermeld staat, zijn globaal. Voor dit soort functies en procedures mag in het implementatie-gedeelte de formele parameterlijst achterwege blijven.
- In het initialiserings-gedeelte kunnen die zaken geregeld worden die noodzakelijk zijn om de unit te kunnen gebruiken.
- De unit wordt afgesloten door een END, gevolgd door een punt.

10.3 Het compileren van units

We moeten nu aandacht schenken aan de programmeeromgeving. Een unit is geen volledig programma en kan daarom niet zelfstandig uitgevoerd worden. Als units gecompileerd worden, dan worden ze door de compiler op schijf gezet met de extensie .TPU (van Turbo Pascal Unit).

Als je in de programmeeromgeving kiest voor de opties «*Compile*» en «*Compile*» of «*Alt-F9*», dan wordt het aangeboden bestand gecompileerd. Een programma wordt gecompileerd naar een bestand met de extensie .EXE en een unit naar een bestand met de extensie .TPU, aangenomen dat met de keuzen «*Compile*» en «*Destination*» de bestemming op "Disk" staat en niet op "Memory" (zie eventueel pagina 28).

Bij een keuze voor «*Compile*» en «*Make*», of «*F9*», kijkt de compiler of de units die gebruikt worden sinds de laatste keer dat het programma of de unit gecompileerd is, een verandering hebben ondergaan. In dat geval wordt ook de veranderde unit opnieuw gecompileerd. Dit kan alleen als ook de programmacode van de unit beschikbaar is. Indien deze niet beschikbaar is, onderbreekt de compiler zijn werk met foutmelding 70: "unit version mismatch" (verkeerde unitversie).

De keuze voor «*Compile*» en «*Build*» compileert het aangeboden programma of de unit. Ook worden alle units die in de USES-lijst staan, en waarvan de programmacode beschikbaar is, gecompileerd.

Je kunt ervoor zorgen dat de compiler bij een keuze voor «*Make*» of «*Build*» niet de unit waaraan je misschien toevallig aan het werk bent als uitgangspunt neemt, maar het programma waar de unit deel van uit maakt. In dat geval kies je in het menu voor «*Compile*» en «*Primary File*», en kies je vervolgens uit de aangeboden bestandenlijst het bestand dat het programma bevat.

Het gebruik van units werkt in Turbo Pascal erg efficiënt voor wat betreft de geheugenruimte die in beslag wordt genomen. De compiler verwerkt namelijk alleen de functies en procedures die ook daadwerkelijk in het programma gebruikt worden.

10.4 Cirkelverwijzing

Units kunnen via hun USES-regel andere units gebruiken. Deze methode heeft wel een beperking. Stel dat je beschikt over een unit A. Deze unit heeft in zijn USES-regel de opdracht staan: "USES B". Unit B heeft in zijn USES-regel staan: "USES A". In dat geval onstaat er een cirkelaanroep. Als unit A gecompileerd wordt, moeten er zaken uit unit B gehaald worden. De compiler begint met het compileren van unit B. Maar helaas. Als de compiler aan unit B begint, vraagt deze om unit A, want dat staat in zijn USES-regel. Unit A was echter nog niet klaar en vraagt om zaken uit unit B. Zo blijven we aan de gang. Deze fout wordt genadeloos afgestraft met een foutmelding 68: "circular reference" (cirkelverwijzing).

Het volgende voorbeeld laat zien wat er in de praktijk gebeurt als deze fout gemaakt wordt. Als je Turbo Pascal gebruikt, moet onder de menukeuze «*Compile*» de optie «*Destination*» op "Disk" staan (zie eventueel pagina 28):

```
UNIT UNIT_A;
Interface
USES UNIT_B;

PROCEDURE Identificatie;
Implementation
```

```
PROCEDURE Identificatie;
   BEGIN
      Writeln('Dit is UNIT_A')
   END
END.

UNIT UNIT_B;
Interface
USES UNIT_A;

PROCEDURE Identificatie;
Implementation

PROCEDURE Identificatie;
   BEGIN
     Writeln('Dit is UNIT_B.')
   END
END.
```

We zien hier twee zeer eenvoudige units die beide slechts één globale procedure bevatten. Dit programma zal door de compiler geweigerd worden, omdat UNIT_A in de USES-regel een verwijzing naar UNIT_B heeft staan terwijl UNIT_B aangeeft dat UNIT_A moet worden gebruikt.

Het volgende programma geeft dan ook een foutmelding:

```
PROGRAM UNIT_2;
USES UNIT_A, UNIT_B;
BEGIN
   UNIT_A.Identificatie;
   UNIT_B.Identificatie
END.
```

Aan deze cirkelverwijzing valt alleen te ontkomen als beide units een USES-regel opnemen onder het beschermde woord "implementation". In dat geval maken we het gebruik van de aangeroepen unit lokaal en blijft een foutmelding achterwege. Het volgende programmaatje demonstreert dit:

```
UNIT UNIT_C;
Interface
PROCEDURE Identificatie;

Implementation
USES UNIT_D;

PROCEDURE Identificatie;
   BEGIN
      Writeln('Dit is UNIT_C.')
   END;
```

```
END.


UNIT UNIT_D;
Interface
PROCEDURE Identificatie;

Implementation
USES UNIT_C;

PROCEDURE Identificatie;
   BEGIN
      Writeln('Dit is UNIT_D.')
   END;
END.
```

Het volgende programma zal keurig uitgevoerd worden zonder een foutmelding van de compiler. Omdat beide units een procedure bevatten met dezelfde naam, zou het zonder meer aanroepen van Identificatie steeds de procedure van UNIT_D aanroepen. Deze unit staat als laatste in de USES-regel van het programma. In zo'n geval kun je de juiste procedure krijgen door eerst de naam van de unit te geven, gevolgd door een punt en tenslotte de naam van de aan te roepen procedure. Deze manier van lezen komt overeen met het lezen van een record:

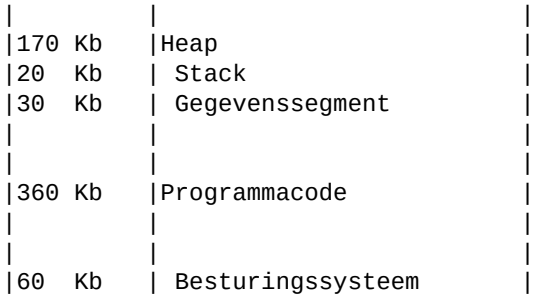
```
PROGRAM UNIT_3;
USES UNIT_C, UNIT_D;
BEGIN
   UNIT_C.Identificatie;
   UNIT_D.Identificatie;
   Readln
END.
```

10.5 Overlays

Overlays zijn delen van een programma die afwisselend hetzelfde deel van het geheugen gebruiken. Om overlays te begrijpen zullen we even in de geheugenorganisatie van een Turbo Pascal-programma moeten duiken. De compiler van Turbo Pascal werkt met een intern geheugenmodel van maximaal 640 Kb geheugenruimte. De compiler van Borland Pascal kan met veel meer geheugen werken. Als de compiler met het maximum van 640 Kb werkt, dan spreekt men wel van werken in de "real mode". Werkt een compiler met het uitgebreide geheugen (extended memory), dan spreken we van "protected mode".

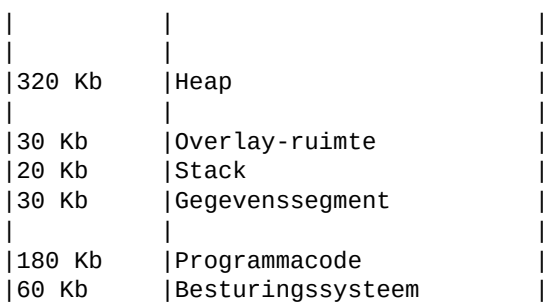
Voor de bespreking van overlays nemen we het 640 Kb intern geheugenmodel van Turbo Pascal als uitgangspunt. Het besturingssysteem legt in de eerste plaats beslag op een stukje van het geheugen. Een programma is een verzameling instructies voor de computer. Als het programma uitgevoerd moet worden, moet dit in het geheugen opgeslagen worden. Daarnaast werkt het programma met gegevens. Een deel van het geheugen (het datasegment) zal

gereserveerd worden voor de opslag van deze gegevens. Om binnen een programma sprongen te kunnen maken, is er een plaats nodig waar gegevens tijdelijk bewaard kunnen worden. Deze tijdelijke opslag vindt plaats in het stacksegment. De programmeur stelt bij het compileren de grootte van de stack in. De rest van het geheugen (de heap) is gereserveerd voor overige gegevens. Schematisch zou de opslag van een programma in het geheugen er zo uit kunnen zien:



Het gaat hier kennelijk om een groot programma. Ook zien we dat de programmacode een relatief groot deel van het geheugen in beslag neemt. Er blijft maar weinig ruimte over voor de heap. Het zou best kunnen dat het functioneren van het programma in gevaar komt omdat er maar zo weinig ruimte op de heap beschikbaar is.

Turbo Pascal beschikt over de mogelijkheid om het beslag op het geheugen door het programma te verminderen. Hierbij krijgt een aantal delen van het programma dezelfde geheugenruimte ter beschikking. Een deel van het programma wordt in het geheugen ingelezen als het nodig is. Is dat onderdeel op dat moment niet nodig voor de programma-uitvoering, dan blijft het gewoon op de schijf staan. We maken in deze situatie gebruik van zogenaamde overlays. Het schema geeft aan dat de programmacode 360 Kb in beslag neemt. Stel dat we van 360 Kb een ruimte van 180 Kb afhalen en dit verdelen in zes gelijke porties. Dan hebben we een programma van 180 Kb en zes deelprogramma's van 30 Kb op schijf. Het programma leest een deel in op het moment dat dit voor de uitvoering van het programma nodig is. Deze delen zijn de overlays. De verdeling van het geheugen krijgt nu een heel ander aanzien:



Uit dit schema blijkt dat we met deze handelwijze 150 Kb aan geheugencapaciteit gewonnen hebben. Bij het gebruik van overlays reserveert de compiler net zoveel geheugenruimte als nodig is om

de grootste overlay in op te kunnen slaan. De ruimte die gereserveerd wordt, wordt de overlay-buffer genoemd.

Overlays kunnen elkaar aanroepen. Je moet er dan wel rekening mee houden dat het programma trager wordt. Als Overlay 1 een functie van Overlay 2 aanroept, moet eerst Overlay 2 van schijf gehaald en in het geheugen geplaatst worden. Nadat de functie afgewerkt is, moet er teruggekeerd worden naar Overlay 1. Deze bevindt zich niet meer in het geheugen. Op die plaats is immers Overlay 2 ingelezen. Dus moet Overlay 1 weer van schijf gehaald worden en in het geheugen worden ingelezen.

Als de overlays van verschillende grootte zijn, dan houdt de overlay-manager zoveel mogelijk overlays in het geheugen. De overlay-manager wordt geleverd door de unit OVERLAY. Het is deze manager die de afmeting van de overlay-buffer bepaalt. Deze buffer wordt dan net zo groot als de afmeting van de grootste overlay.

De overlay-buffer is echter niet de baas, dat is de programmeur. De programmeur kan zelf beslissen of hij of zij het nodig vindt een grotere buffer te gebruiken waar verschillende overlays tegelijk in kunnen. Als je dit wilt, kun je de functie OvrSetBuf gebruiken. De functie OvrSetBuf krijgt een parameter mee waarin de afmeting van de overlay-buffer moet staan. De functie OvrGetBuf retourneert de afmeting van de buffer, en met de functie OvrClearBuf wordt de overlay-buffer leeggemaakt. In de variabele OvrResult wordt gerapporteerd of de operaties goed verlopen zijn. Onderstaande tabel geeft een overzicht van de overlay-meldingen:

Constante:	Waarde:	Betekenis:
OvrOk	0	Operatie succesvol.
OvrError	-1	Fout bij de overlay-manager.
OvrNotFound	-2	Overlay-bestand niet gevonden.
OvrNoMemory	-3	Niet genoeg geheugen voor overlay-buffer.
OvrIOError	-4	Leesfout in overlay-bestand.
OvrNoEMSDriver	-5	EMS-driver niet geïnstalleerd.
OvrNoEMSMemory	-6	Niet genoeg EMS-geheugen.

De letters "EMS" staan voor "Expanded Memory Specification". Dit is een uitbreiding van het werkgeheugen van een computer. Om EMS-geheugen te kunnen gebruiken, heb je een programma (de EMS-driver) nodig dat dit geheugen bestuurt.

Als het aanroepen van overlays een enkele keer voorkomt, zal dit nauwelijks effect hebben op de snelheid van je programma. Als dit echter herhaaldelijk gebeurt, bijvoorbeeld vanuit een lus, dan moet er nogal wat gelezen en geladen worden. Het programma zal dan trager worden. De oplossing in dat geval is om één van de overlays weer deel uit te laten maken van de gewone programmacode. Overigens is het verstandig om veelvuldig gebruikte procedures en functies niet in een overlay te zetten.

Om te laten zien hoe overlays werken nemen we weer twee kleine

units in een klein programma op:

```
{$O+, $F+}
UNIT UNIT_E;
Interface

PROCEDURE Identificatie;
Implementation
USES UNIT_F;

PROCEDURE Identificatie;
   BEGIN
      Writeln('Dit is de overlay UNIT_E.')
   END;
END.

{$O+, $F+}
UNIT UNIT_F;
Interface

PROCEDURE Identificatie;
Implementation
USES UNIT_E;

PROCEDURE Identificatie;
   BEGIN
      Writeln('Dit is overlay UNIT_D.')
   END;
END.
```

Als je het volgende programma draait, worden de units UNIT_E en UNIT_F verwerkt als overlays. Het concept van overlays is dat alleen units als overlays kunnen functioneren:

```
PROGRAM UNIT_4;
USES OVERLAY, UNIT_E, UNIT_F;
{$O UNIT_E}
{$O UNIT_F}

BEGIN
   OvrInit('UNIT_4.OVR');
   UNIT_E.Identificatie;
   UNIT_F.Identificatie;
   Readln
END.
```

Om een unit als overlay te kunnen laten functioneren, moet in de eerste plaats boven de aanduiding UNIT

de compilerinstructie {$O+} gezet worden. Deze instructie geeft de compiler het recht om de unit als overlay te verwerken. Naast de instructie {$O+} moet ook de instructie {$F+} gezet worden. Met deze laatste instructie "aan", genereert Turbo Pascal uitgebreide geheugenadressen. Voor het werken met overlays zijn die noodzakelijk. Als je de instructie {$F+} vergeet kan het programma rare kuren gaan vertonen.

In de USES-regel van het programma moet de unit OVERLAY opgenomen worden. Het is deze unit die ervoor zorgt dat het proces van indelen en oproepen van overlays goed verloopt. Tenslotte moet op de eerste regel van het hoofdprogramma de procedure OvrInit worden aangeroepen. OvrInit is een in de unit OVERLAY gedefinieerde procedure. Als parameter krijgt hij een string mee. Deze string bevat een bestandsnaam. De verschillende overlays worden in een bestand met deze bestandsnaam geschreven. Let erop dat de bestandsnaam dezelfde naam heeft als het programma, met toevoeging van de extensie .OVR.

Na de naam van het programma en de USES-lijst moet je aangegeven welke units daadwerkelijk als overlay deel zullen gaan uitmaken van het programma. De opdrachten:

```
     {$O UNIT_E}
{$O UNIT_F}
```

zorgen ervoor dat de gecompileerde code van de units UNIT_E en UNIT_F als overlay geplaatst wordt in het bestand UNIT_4.OVR. Het programma zal de units nu als overlay behandelen.

Als een unit als overlay gebruikt wordt, dan kan in principe geen gebruik gemaakt worden van het initialiserings-gedeelte van de unit. Normaal gesproken is het onmogelijk om op die manier variabelen van de units te initialiseren. Dat het initialiserings-gedeelte niet gebruikt kan worden, is eigenlijk wel logisch. De overlay-units worden beheerd door de overlay-manager. De initialisering van de units vindt plaats voordat de eerste regel van het programma uitgevoerd wordt, waar een aanroep van OvrInit staat. Deze aanroep initialiseert de overlay-manager. De consequentie hiervan is dat overlay-units geïnitialiseerd zouden moeten worden voordat de overlay-manager geïnitialiseerd is.

Er is een manier om uit de impasse te komen. Je moet dan een lege unit maken, die geen andere functie heeft dan het in het initialiserings-gedeelte aanroepen van OvrInit. Deze unit moet je dan vooraan in de USES-lijst aanroepen, zodat dit de eerste initialisering is die wordt uitgevoerd.

10.6 Opgaven

10.1 Maak een samenvatting van de regels waar je je aan moet houden om overlays te kunnen gebruiken.

183

183

183

197

197

197