

Die „Interactive Data Language“ *IDL* der Firma Research Systems Inc., Boulder, USA ist ein Standard Software-System für die Analyse, Reduktion und anspruchsvolle Darstellung (Visualisierung) von wissenschaftlichen Daten.

IDL ist eine Sprache der 4. Generation, die es jedem Wissenschaftler ermöglicht, selbst seine Daten interaktiv schnell und genau auf verschiedenen Computer-Plattformen zu analysieren. *IDL* ist auch eine vollständige Programmiersprache, die ein

strukturiertes Programmieren erlaubt.

In diesem Merkblatt sollen die verschiedenen Strukturen, die *IDL* bietet, genauer dargestellt werden, zumal die Beschreibungen im *IDL*-Handbuch [1] etwas sehr knapp sind.

1. Die Programm-Module:

Auch unter *IDL* können Programme in verschiedenenartigen Modulen realisiert werden. Dieses sind Hauptprogramme, Unterprogramme (Prozeduren und Funktionen), Batch-Programme sowie Kombinationen davon.

In den Abb. 1 – 5 sind die Grundstrukturen dieser Module dargestellt. Jedes dieser Module ist gleichzeitig eine eigenständige Datei (file). Bei Prozeduren und Funktionen sollte die Dateibezeichnung dem jeweiligen Namen entsprechen.

■ **Hauptprogramme** (main programs) haben am Anfang keine besondere Kopf-Anweisung (Abb. 1). Sie bestehen nur aus *IDL*-Anweisungen (statements) und müssen mit einer *END*-Anweisung abgeschlossen werden.

Ein Hauptprogramm kann nicht von anderen Modulen aufgerufen werden. Es können keine Parameter übergeben werden. Es wird mit dem Befehl *.RUN* geladen und gestartet [2]. Die bei einer *IDL*-Sitzung automatisch erzeugte Protokoll-Datei *IDL_SAVE.PRO* ist ein solches Hauptprogramm.

■ **Prozeduren** (procedures) sind unter *IDL* abgekapselte Unterprogramm-Module in besonderen Dateien (Abb. 2) oder innerhalb der Datei eines Hauptprogramms (Abb. 4). Die Kopf-Anweisung muß „*PRO ...*“ sein. Prozeduren werden mit einer *END*-Anweisung abgeschlossen.

Eine Prozedur kann von anderen Modulen aufgerufen werden. Es können dabei Parameter in beiden Richtungen übergeben werden. Außerdem ist eine Parameterübergabe per Common-Bereich – wie bei Fortran – möglich.

■ **Funktionen** (functions) sind ebenfalls gekapselte Unterprogramm-Module mit allen Eigenschaften der Prozeduren. Die Kopf-Anweisung muß „*FUNCTION ...*“ sein (Abb. 3).

Eine *IDL*-Funktion übergibt mit der *RETURN*-Anweisung einen Funktionswert an das rufende Programm. Unter *IDL* ist der Funktionswert nicht auf skalare Datentypen beschränkt. Es können komplette Felder (arrays) oder (komplexe) Datenstrukturen (structures) übergeben werden. Eine Funktion muß eine, sie darf mehrere *RETURN*-Anweisungen enthalten.

■ **Batch-Programme** (batch files) dienen zum nicht-interaktiven (automatischen) Ablauf von *IDL*-Anweisungen. Jede Zeile eines *IDL* Batch-Programms wird dabei so ausgeführt, als wenn er interaktiv über das Tastenfeld eingegeben würde.

Ein Batch-Programm wird unter *IDL* mit *@IDL_BatchJob* gestartet. *IDL*-Anweisungen, die mehrere Zeilen benötigen (z. B. eine *REPEAT*-Schleife), müssen am Zeilenende jeweils mit dem „\$“ versehen werden (Abb. 5).

Die *IDL* Batch-Programme können auch direkt von der Betriebssystemebene gestartet werden, mit z. B. unter *VMS*:

```
VAXSER> IDL IDL_BatchJob.PRO
```

Damit ist es möglich *VMS* Batch-Jobs anzufertigen, die *IDL*-Programme unbeaufsichtigt ablaufen lassen können [3].

```
IDL-Statements
...
...
END ; Dieses „End“ darf nicht fehlen.
```

Abb. 1: Struktur eines *IDL*-Hauptprogramms (main).

```
PRO ProzName, Param1, Param2, ...
...
IDL-Statements
...
...
RETURN ; Dieses „Return“ ist optional.
END ; Dieses „End“ darf nicht fehlen.
```

Abb. 2: Struktur einer *IDL*-Prozedur (Unterprogramm).

```
FUNCTION FunkName, Param1, Param1, ...
...
IDL-Statements
...
...
RETURN, FunktionsWert ; Darf nicht fehlen.
END ; Dieses „End“ darf nicht fehlen.
```

Abb. 3: Struktur einer *IDL*-Funktion (Unterprogramm).

```
PRO ProzName, Param1, Param2, ...
...
IDL-Statements
...
...
RETURN ; Dieses „Return“ ist optional.
END ; Dieses „End“ darf nicht fehlen.

FUNCTION FunkName, Param1, Param1, ...
...
IDL-Statements
...
...
RETURN, FunktionsWert ; Darf nicht fehlen.
END ; Dieses „End“ darf nicht fehlen.

IDL-Statements
...
...
END ; Dieses „End“ darf nicht fehlen.
```

Abb. 4: Struktur eines *IDL*-Hauptprogramms (unten) mit eingebauten Prozeduren (oben).

```
IDL-Statements ; Auf einer Zeile.
...
...
IDL-Statement $ ; Ein IDL-Statement,
IDL-Statement $ ; das über mehrere
IDL-Statement ; Zeilen läuft.
...
; Hier darf kein „End“ stehen!
```

Abb. 5: Struktur einer *IDL*-Batchdatei.

2. Die Schleifen:

Unter *IDL* stehen vier Arten von Schleifen zur Verfügung, die WHILE-, die REPEAT-, die FOR- und die sogenannte „Label-GOTO“-Schleife.

■ Mit **WHILE-Schleifen** werden Anweisungsfolgen solange wiederholt bis eine bestimmte Bedingung (condition) erfüllt ist.

```
WHILE ConditionExpression
  BEGIN
    ...
    IDL-Statements
    ...
  ENDWHILE
```

Abb. 6: Struktur einer WHILE-Schleife.

Bei einer WHILE-Schleife wird zu Beginn die Schleifenbedingung ausgewertet. Wenn sie erfüllt ist, wird die zwischen BEGIN und ENDWHILE stehende Anweisungsfolge (der Rumpf der Schleife) ausgeführt und die WHILE-Schleife erneut ausgeführt. Ist die Schleifenbedingung nicht erfüllt, wird die Verarbeitung sofort hinter dem ENDWHILE fortgesetzt.

Da die Prüfung der Schleifenbedingung immer vor der Ausführung des Rumpfes erfolgt, kann der Fall eintreten, daß die WHILE-Schleife überhaupt nicht ausgeführt wird. Man bezeichnet sie deshalb auch als *Abweisschleife*.

■ **REPEAT-Schleifen** ermöglichen die Wiederholung einer Anweisungsfolge, bis eine bestimmte Bedingung erfüllt ist.

```
REPEAT BEGIN
  ...
  IDL-Statements
  ...
ENDREP UNTIL ConditionExpression
```

Abb. 7: Struktur einer REPEAT-Schleife.

Zuerst wird die Anweisungsfolge zwischen REPEAT und ENDREP ausgeführt. Anschließend wird die Schleifenbedingung hinter dem Schlüsselwort UNTIL ausgewertet. Ist die Bedingung erfüllt, wird die Verarbeitung mit der auf die REPEAT-Schleife folgenden Anweisung fortgesetzt, sonst wird die REPEAT-Schleife erneut ausgeführt.

Die Schleifenbedingung wird immer erst nach der Ausführung der Anweisungsfolge geprüft. Daher wird der Rumpf einer REPEAT-Schleife mindestens einmal ausgeführt. Sie heißt deshalb auch *Durchlaufschleife*.

■ **FOR-Schleifen** entsprechen den von Fortran bekannten DO-Schleifen. Sie werden zur Wiederholung einer Anweisungsfolge mit einer festen Anzahl von Durchläufen verwendet. Die FOR-Schleife ist wie die WHILE-Schleife eine *Abweisschleife*.

```
FOR Var = Expr1, Expr2, [Expr3] DO BEGIN
  ...
  IDL-Statements
  ...
ENDFOR
```

Abb. 8: Struktur einer FOR-Schleife.

Der Rumpf einer FOR-Schleife wird mehrmals ausgeführt, wobei die durch *Var* bezeichnete Laufvariable jedesmal einen anderen Wert annimmt. Der Ausdruck *Expr1* bestimmt den Anfangswert, der Ausdruck *Expr2* bestimmt das Ende des überstrichenen Wertebereichs. Mit dem Ausdruck *Expr3* wird die Schrittweite – die positiv oder negativ sein kann – festgelegt, um die der Wert der Laufvariablen nach jedem Schleifen-

durchlauf erhöht (bzw. vermindert) wird. Fehlt die Angabe von *Expr3* wird die Schrittweite + 1 genommen. Es ist ein sehr schlechter Programmierstil, den Wert der Laufvariablen *Var* innerhalb des Rumpfes zu verändern!

■ **„Label-GOTO“-Schleifen** sollten – als Eigenkonstrukte – nach Möglichkeit vermieden werden. Hier ein Beispiel:

```
Label:
  ...
  IDL-Statements
  ...
  IF ConditionExpr THEN GOTO Label
```

Abb. 9: Struktur einer „Label-GOTO“-Schleife.

3. Die CASE-Anweisung:

Mit der CASE-Anweisungen können Verzweigungen mit vielen Fällen, die nur von verschiedenen Werten *eines* Ausdrucks abhängen, kurz und klar formuliert und schneller ausgeführt werden.

```
CASE Expression OF
  Expr1: BEGIN
    ...
    IDL-Statements
    ...
  END
  Expr2: ...
  ...
  ELSE: IDL-Statement
ENDCASE
```

Abb. 10: Struktur einer CASE-Anweisung.

4. Die IF-Anweisung:

Die IF-Anweisung dient zur Verzweigung in Abhängigkeit von Bedingungen.

```
IF ConditionExpr $
  THEN BEGIN
    ...
    IDL-Statements
    ...
  ENDIF $
  ELSE BEGIN
    ...
    IDL-Statements
    ...
  ENDELSE
```

Abb. 11: Struktur einer IF-Anweisung.

5. Sonstige Hinweise:

■ Aufgrund eines „Bugs“ (Fehlers) im derzeitigen IDL, enthält die Protokoll-Datei IDL_SAVE.PRO am Schluß keine END-Anweisung. Daher muß diese ggf. mit einem Editor zugefügt werden, sonst ist dieses Hauptprogramm nicht lauffähig.

6. Literatur:

- [1] IDL – Interactive Data Language. Reference Guide. Version 3.0. Boulder (USA): Research Systems Inc., Januar 1993. E-Mail: idl@boulder.colorado.edu.
- [2] Dittberner, K.-H.: Interactive Data Language IDL: Eine Einführung. FU Berlin (IfP): wdv-notes Nr. 85, 1988–1994.
- [3] Dittberner, K.-H.: IDL-Programme für die Signaldatenverarbeitung. FU Berlin (IfP): wdv-notes Nr. 2, 1988–1995.

Notizen: