GRUNDLAGEN WISSENSCHAFTLICHEN ARBEITENS

TUTORIAL ON MODULA-2
(WINTERSEMESTER 2004/2005)

AUTOR

NACHNAME :SUBAYAZ
VORNAME :SONAY
MATRIKELNUMMER:0027794
KENNZAHL :535
(e0027794@student.tuwien.ac.at)

LEITER:PROF.DR.JENS KNOOP

KURZFASSUNG

Modula-2 ist der Nachfolger von Pascal und wurde Ende der 1970er Jahre durch N. Wirth entwickelt. In Modula -2 ein weiteres Konzept der Strukturierten Programmierung zur Verfügung gestellt, und zwar wird durch eine Modularisierung die bisher noch vorhandene monolithische Struktur eines Programms aufgebrochen.

GRUNDLAGEN

Modula-2 ist eine universelle Programmiersprache, die vor allem dafür konzipiert wurde, Systemprogramme zu schreiben. Der Name Modula ist eine Abkürzung für „**modu**lar **la**nguage“.
Modula-2 ist die dritte in einer Reihe von Sprachen, die von dem Schweizer Informatiker Niklaus Wirth geschaffen wurden. Die erste Sprache, Pascal, war als Ausbildungssprache gedacht, wird heute aber auf dem verschiedensten Gebieten verwendet. Die zweite, Modula, war eine Spezialsprache, um kleine Echtzeitsysteme zu programmieren.Modula-2 vereinigt die Möglichkeiten zur Systemimplementierung von Modula und die allgemeine Anwendbarkeit von Pascal.

Modula-2 weist die meisten Merkmale von Pascal auf, weicht aber in drei Richtungen davon ab:

- Es erweitert Pascal „nach oben“, um die Systemprogrammieren einzuschließen. In Modula-2 können große Softwaresysteme programmiert werden, ohne dass sie von einem Betriebsystem unterstützt werden müssen.
- Es erweitert Pascal „nach unten“, um Programmierung auf Maschinenebene zu ermöglichen. Durch Modula-2 werden auf den untersten Niveaus eines Rechnersystem Assemblersprachen überflüssig.
- Es führt kleinere Änderungen im Vergleich zu Pascal ein, die das Programmieren vereinfachen und die Lesbarkeit und Effizienz der Programme verbessern.

Modula-2 baut, wie der Name schon sagt, auf dem **Modelkonzept** auf. Ein Modul ist eine Gruppe zusammenhängender Prozeduren und Daten, die zusammen als **Objekte** bezeichnet werden. Ein Modul erlaubt den Zugriff auf einige seiner Objekte von außerhalb, verbirgt aber die Existenz anderer Objekte vor dem Rest des Programms.

Obwohl sie konzeptuell einfach sind, haben Module ein überraschend breites Anwendungsfeld:

- Module gestatten die Strukturierung großer Programme auf eine besser lesbare Art, als sie bei Sprachen wie Pascal möglich ist: kleine Sammlungen von Modulen (die nur relativ wenige Objekte gemeinsam haben) ersetzen die herkömmliche Ansammlung von Prozeduren , die durch seitenlange globale Vereinbarungen miteinander verknüpft sind.
- Die Möglichkeit, durch Module ein Programm in wohldefinierte Teile zu gliedern, bildet eine Grundlage für die getrennte von einem Übersetzbarkeit. Modula-2 definiert eine spezielle Art von Modulen , die getrennte von einem Programm übersetzt werden. Große Programme können in einzelne, getrennt übersetzbare Module aufgeteilt werden; andererseits können getrennt übersetzte Module in einer Modulbibliothek gespeichert und den verschiedensten Programmen verwendet werden.
- Die Möglichkeit, durch Module Objekte vor der Außenwelt zu verbergen, erlaubt es, portable Programme zu schreiben, die maschinennahe Teile enthalten. Der Bezug auf Eigenschaften einer speziellen Maschine kann auf Module beschränkt werden und hat dadurch nur für kleiner Programmteile Konsequenzen. Diese Module haben eine maschineunabhängige Schnittstelle - der direkte Zugriff auf maschineabhängige Objekte wird dadurch verhindert. Wenn Programme auf verschiedene Rechner transferiert werden, bleibt also der Großteil der Software unverändert; nur die maschineabhängigen Module müssen neu geschrieben werden.
- Die Möglichkeit , durch Module Datentyp zu definieren, die ihre interne Struktur verbergen, fördert die Verwendung eines Programmierkonzepts, das unter der Bezeichnung **abstrakte Datentypen** bekannt ist. Alle Operationen auf abstrakten Datentypen werden durch Prozeduraufrufe ausgeführt; auf ihre Daten kann nicht direkt zugegriffen werden. Ein bekanntes Beispiel eines abstrakten Datentyp ist der in Pascal vordefinierte Datentyp; Pascal ist jedoch nicht in Lage, neue abstrakte Datentypen zu definieren

.Kurzüberbilck Syntax und Semantik

Die Beschreibung einer Programmiersprache geschieht auf der Basis eines Alphabets (Zeichenvorrat) durch die Syntax und die Semantik. Als Alphabets werden die üblichen Zeichen (Buchstaben ohne Umlaute, Ziffern, Satzzeichen, Leerzeichen,…)verwendet.

Die Syntax beschreibt die Regeln, nach denen aus diesen Zeichen ein gültiges Programm geformt wird. Wir beschreiben die Syntax von Modula-2- Programmen durch Syntaxdiagramme. Eine andere gebräuchliche Form der Beschreibung ist die erweiterte Backus-Naur-Forn.

Jedem gültigen Programmkonstrukt wird durch die Semantik eine Bedeutung zugeordnet. Wir geben für die Semantik keine formale Beschreibung, sondern Regeln in umgangssprachlicher Form an.

Für jedes Programmkonstrukt gibt es ein eigenes Syntaxdiagramme. Diese Diagramme werden ineinander eingesetzt, um so ein komplettes Programm zu beschreiben.

Ein Programm -in Modula-2 als *Programmmodul* bezeichnet – wird durch das Wort MODULE (als erstes Wort) als solches gekennzeichnet. Es hat eine *Namen* und besteht im wesentlichen aus einem *Block*, der nach dem Namen – von diesem durch Semikolon (;) getrennt – folgt. Das Programm wird durch seinen Namen gefolgt von einem Punkt (.) abgeschlossen. Vor dem Block können (bzw. müssen) in einem oder mehreren *Import*-Teilen die aus anderen Modulen importierten Objekte angegeben werden. Nach dem Modulnamen, vor dem ersten Semikolon, kann eine Priorität – durch eine ganze Zahl, die wir als *Konst A* bezeichnen wollen und die in eckige Klammern([bzw.]) eingeschlossen ist .

Programmieren

Aufbau eines Modula-2 Programms

Das einfachste Programm soll einen kurzen Text auf dem Bildschirm ausgeben. An diesem Beispiel kann der allgemeine Aufbau eines Modula-2 Programmes dargelegt werden:

MODULE prog1;

```
FROM InOut IMPORT WriteString;
BEGIN
   WriteString ('hello, world');
END prog1.
```

Das Programm prog1 besteht aus drei Teilen:

Der Programmkopf besteht aus dem Schlüsselwort MODULE gefolgt von einem Bezeichner als dem Namen des Programms; der Bezeichner muß mit einem Buchstaben beginnen und kann mehrere Buchstaben und Zahlen enthalten. Am Zeilenende steht ein Semikolon.

In Modula2 sind Ausgabeanweisungen nicht ohne weiteres verfügbar; daher muß der Programmierer im Vereinbarungsteil die Anweisung WriteString ausdrücklich aus dem Standardmodul InOut anfordern. Am Zeilenende steht wieder ein Semikolon als Abschluß.

Schließlich enthält der mit dem Schlüsselwort BEGIN anfangende Ausführungsteil diejenigen Befehle, die bei dem Aufruf des Programms bearbeitet werden sollen. WriteString ('Text'); bedeutet "gib den Text zwischen den Hochkommas aus". Nach jedem Befehl muß ein Semikolon stehen. Als Kennzeichnung von dem Programmende steht das Schlüsselwort END gefolgt von dem Programmnamen und einem Punkt.

Deklarationen

• Modula-2 kennt sechs „elementare“ Typen, die folgende

(reservierten) Bezeichner tragen:

INTEGER, CARDINAL, REAL,

BOOLEAN, CHAR, BITSET

• das Typkonzept von Modula-2 erlaubt es, eigene Typen den

eingebauten Typen hinzuzufügen (durch eine Typdeklaration)

TYPEN

INTEGER

• ganze Zahlen innerhalb des auf dem Rechner

darstellbaren Wertebereichs

• 16-Bit-Rechner bieten Werte zwischen -32768 und 32767

• Operatoren: +,-,*,DIV,MOD,=,<>,<,>,<=,>=

CARDINAL

• positive ganze Zahlen mit rechnerabhängigem Wertebereich

• 16-Bit-Rechner bieten Werte zwischen 0 und 65535

• Operatoren analog zu INTEGER

REAL

• gebrochene Zahlen (Dezimalbrüche) mit einem vom

Rechnertyp abhängigen Wertebereich

• Operatoren: +,-,*,/,=,<>,<,>,<=,>=

• unvollkommene Repräsentation im Rechner führt zu

Rechenungenauigkeiten.

z.B. (1/3)*3 ergibt den Wert 0.9999999

BOOLEAN

• Wahrheitswerte (boolesche Werte)

• Wertebereich: TRUE und FALSE

• Operatoren: AND, OR, NOT, <, >

CHAR

• Zeichen des auf dem jeweiligen Rechner verfügbaren

Zeichensatzes

• Jedem Zeichen ist eine Ordinalzahl zugeordnet, die beim

Vergleichen von CHARs verwendet wird

• Die Buchstaben und Ziffern haben aufeinanderfolgende

Ordinalzahlen (vgl. ASCII-Tabelle).

Beispiel.: Die Frage „Ist ein Objekt c des Types CHAR eine

Ziffer?“ kann in Modula-2 formuliert werden durch:

(c>= "0") AND (c <= "9")

Schlüsselwörter

- Modula-2 kennt einige Wörter mit fester Bedeutung, die sogenannten

Schlüsselwörter (Key Words)

• Diese besitzen eine definierte, nicht änderbare Semantik (Bedeutung).

• Schlüsselwörter sind reserviert und dürfen nicht als Bezeichner für

(benutzer)eigene Module, Prozeduren, etc. verwendet werden.

• Schlüsselwörter müssen immer groß geschrieben werden.

```
AND            ELSIF          LOOP        REPEAT
ARRAY          END            MOD         RETURN
BEGIN          EXIT           MODULE      SET
BY             EXPORT         NOT         THEN
CASE           FOR            OF          TO
CONST          FROM           OR          TYPE
DEFINITION     IF             POINTER     UNTIL
DIV           IMPLEMENTATION  PROCEDURE   VAR
DO            IMPORT          QUALIFIED   WHILE
ELSE          IN              RECORD       WITH
```

Fallunterscheidung

- Die IF -Anweisungsfolge

Bei der Verwendung der IF -Anweisung sind folgende Regeln zu beachten:

- Auf die Anweisungsfolge hinter THEN darf beliebig oft das Schlüsselwort ELSIF (wiederum gefolgt von Ausdruck, THEN, Anweinungsfolge) angegeben werden.

- Nach dem IF/THEN – Zweig bzw. dem letzten ELSIF kann das Schlüsselwort ELSE, gefolgt von einer Anweisungsfolge angegeben werden.

- Das Ergebnis jedes Ausdrucks nach IF oder ELSIF muss vom Typ BOOLEAN sein.

- Der Ausdrucks vom Typ BOOLEAN hinter IF wird ausgewertet. Ist der Wert TRUE, so wird die Anweisungsfolge hinter THEN ausgeführt. Ist der Wert FALSE, so wird, falls ein ELSIF – Zweig vorhanden ist, solange der jeweilige Ausdrucks hinter den folgenden ELSIF ausgewertet, bis einer den Wert TRUE liefert. Die auf dieses ELSIF folgende Anweisungsfolge wird ausgeführt. Liefern alle Ausdrucks den Wert FALSE oder ist kein ELSIF vorhanden, so wird, falls ein ELSE vorhanden ist, diese Anweisungsfolge ausgeführt.

- Die CASE – Anweisungsfolge

Bei Verwendung einer CASE –Anweisung sind folgende Regeln zu beachten:

- Einer CASE – Anweisung beginnt mit dem Schlüsselwort CASE, dem ein Ausdruck und das Schlüsselwort OF folgen.

- Der Ausdruck darf von einem der Datentyp CARDINAL, INTEGER, BOOLEAN, CHAR, Unterbereichstyp oder Aufzählungstyp sein.

- Dem Schlüsselwort OF folg eine Aufzählung aller Fälle, die in Abhängigkeit vom Wert des Ausdrucks ausgeführt werden sollen. Die Fälle werden durch senkrechte Striche voneinander getrennt.

- Jeder „Case“ beginnt mit einer Liste von Konstanten oder Wertebereichen. Jeder darin vorkommende Konstantenausdrucks muss mit dem CASE – Ausdruck ausdruckskompatibel sein. Jeder Wert darf höchstens einmal in derselben CASE – Anweisung vorkommen.

- Im ELSE Zweig können Aktionen angegeben werden, die ausgeführt werden sollen, wenn kein „Case“ eintritt (wenn kein „Case“ eintritt und ELSE nicht vorhanden ist, dann entsteht ein laufzeitfehler).

- Jeder Auswahlmarke (Case Label) und jedem ELSE (das auch wegfallen kann)beliebig viele Anweisungen (auch keine) folgen.

- Die CASE – Anweisung wird mit dem Schlüsselwort END abgeschlossen.

Wiederholungsanweisungen

- Die WHILE – Anweisung

Der vorangestellte boolesche Ausdruck wird ausgewertet. Hat er den Wert TRUE, wird die Anweisungsfolge im Schleifenrumpf abgearbeitet. Nach der letzten Anweisung wird WHILE – Anweisung erneut ausgeführt.

- REPEAT – Anweisung

Die Anweisungsfolge im Schleifenrumpf wird ausgeführt. Anschließend wird der boolesche Ausdruck ausgewertet. Hat dieser den Wert TRUE, so ist die Abarbeitung der Schleife beendet, hat er den Wert FALSE, wird die Schleife erneut abgearbeitet.

- **FOR- Anweisung**

Die Ausdrucke nach := und TO müssen mit der Laufvariablen ausdruckskompatibel sein. Die Schrittweite (positive oder negative Veränderung (=/) des wertes der Laufvariablen) wird durch den Ausdruck nach BY bestimmt; die Angabe der Schrittweite ist optional, Standart ist 1. Die Laufvariable darf im Rumpf verwendet, aber nicht verändert werden.

- **LOOP – Anweisung**

Die LOOP –Anweisung dient dazu, Anweisungen so lange wiederholen zu können, bis eine bestimmte Bedingung erfüllt ist. Der Test, ob diese Bedingung erfüllt ist, kann dabei (auch mehrfach)an beliebigen Stellen in der Schleife erfolgen. Das Verlassen der Schleife geschieht durch die EXIT – Anweisung, die nur innerhalb der LOOP – Anweisungsfolge verwendet werden darf und dort sinnvollerweise mindestens einmal steht.

Wiederholugsanweisungen

Modula kennt vier verschiedene Schleifentypen, die mit den Schlüsselworten REPAT, WHILE, LOOP, und FOR eingeleitet werden.

--Die REPEAT –Schleife

Zu deutsch etwa >>Wiederhole <*Anweisungsfolge*> bis die <*Bedingung*> erfüllt ist<<. Die Bedingung stellt in Modula einen boolschen Ausdruck dar. Die Anweisungsfolge bezeichnet man auch als Schleifenrumpf oder Schleifenkörper.

--Die WHILE – Schleife

Zu deutsch etwa >>solange <*Bedingun*g> erfüllt ist, führe <*Anweisungen*> aus.

Im Gegensatz zur REPEAT –Schleife wird also das Abbruchskriterium am Anfang der Schleife geprüft. Dadurch kann es sein, dass die Schleife nicht ausgeführt wird. Bei der Abbruchsbedingung gibt es noch einen kleinen Unterschied: Die REPEAT –Schleife bricht ab, wenn die Abbruchsbedingung TRUE wird, die WHILE –Schleife, wenn die Abbruchsbedingung FALSE wird (da sie läuft, solange die Bedingung erfüllt ist).

--Die LOOP –Schleife

Das englische Word LOOP heißt direkt übersetzt >>Schleife<<. Dieser Schleifentyp ist bis jetzt noch nicht vorgekommen. Es handelt sich um eine recht allgemeine Schleife. Abbruchsbedingung kann

nämlich irgendwo innerhalb des Anweisungsblocks stehen. Die Anweisung EXIT (engl. EXIT= >>Ausgang<<) beendet die Schleife . In einer LOOP –Schleife sind auch mehrere EXIT –Anweisungen möglich, die Schleife kann also mehrere Abbruchsbedingungen an verschiedenen Stellen haben.

--Die FOR – Schleife

Diese Widerholungsanweisung ist sehr komfortabel, weil die Steuerung der Schleife (Initialisierung. Schrittweite und Abbruchskriterium) bereits im Schleifenkopf vorgegeben sind.

```
FOR i := TO 20 DO

WriteLn;

WriteString(“Hallo”)

END;
```

Hiermit wird genau zwanzigmal der String >>Hallo<< ausgegeben.

Zusammenfassung Zur Erfüllung der Gesamtaufgabe müssen die Module eines Programmsystems untereinander und mit ihrer Umwelt über definierte Schnittstellen kommunizieren. Durch dieses Konzept unterstützt MODULA-2 die Prinzipien der Softwaretechnik

Referenzen

- Puchan/Stucky/Wolf von Gudenberg, Programmieren mit Modula-2,/B.G. Teubner Stuttgart 1991
- Richard Gleaves, Modula-2 / Springer – Verlag Berlin Heidelberg New York Tokyo 1985
- Niklaus Wirth, Programmieren In Modula-2,Springer – Verlag Berlin Heidelberg New York Tokyo 1991
- Weber / Hainer, Programmiersprachen für Mikrocomputer/ B.G. Teubner Stuttgart 1990
- Dürholt. Stefan:MODULA-2 Programmierhandbuch: von der Spracheinführung zur professionellen Soft,1990