Grundlagen Wissenschaftlichen Arbeitens

Institut für Computersprachen
Programmiersprachen und Übersetzer

Thema: Tutorial on Modula-2

Hilal Tekoglu

Matrikelnummer: 0025624 , Studienrichtung: Medizinische Informatik (533)
Email: Hilal_tekoglu@gmx.at

1 Historische Entwicklung und Ziele

Modula-2 ist eine Programmiersprache die von Nikolaus Wirth an der ETH (Eidgenössische Technische Hochschule) Zürich, in der Schweiz Ende der 70 er Jahre entwickelt wurde. Nikolaus Wirth entwickelte neben Modula-2 auch Algol-W, Pascal, Modula und Oberon. Modula gilt als die Weiterentwicklung von Pascal. Wobei Modula-1 nicht die gleiche Popularität erreicht hat wie die objekt-basierte Sprache Modula-2. Modula-2 beseitigt einige Mängel der Sprache Pascal.

Ab dem Jahr 1977 hat das Konzept von einer höheren Programmiersprache, mitunter auch mit Pascal Popularität erlangt. Die Kunst des Programmierens basiert auf das Verständnis der Grundlegendsten Konzepte der Programmiersprache. Deshalb ist es erforderlich eine Notation zu verwenden, welche die Konzepte in einer deutlichen Art und Weise darstellt und den Bedarf an einer Systematischen Struktur betont. [1]

Die Programmiersprache Modula-2 wurde 1978, 10 Jahre nach Pascal, definiert und von L.Geissmann, S.E. Knudsen und C.Jacobi auf dem PDP-11 entwickelt. PDP-11 ist ein 16 Bit Computer. Im Sommer 1979 wurde der Compiler vollendet. Und zur gleichen Zeit wurde der erste Lilith Prototyp betriebsbereit. Lilith ist einer der ersten Personel Computer, welches von Nikolaus Wirth entwickelt wurde. Wie schon bereits erwähnt wurde, ist Modula-2 von Pascal herausgewachsen und berücksichtigt einige Haupt Entwicklungen und einige kleinere Entwicklungen. Modula – 2 erweitert Pascal um das Modul Konzept. Das einzelne bedeutende hinzugefügte Einrichtung ist die Modul Struktur. Diese Eigenschaft erlaubt dem Programmierer die deklarierten Objekte zu kontrollieren, ob sie sichtbar oder versteckt sein sollen. Man kann sagen, dass diese Eigenschaft einer der Wichtigsten und bedeutendsten Ziel war. Obwohl dieses Prinzip von „Information Hiding“ in den frühen 70er

Jahren diskutiert wurde, ist es ein Verdienst von Modula, dass dieses Prinzip übereinstimmend in das Framework einer klar definierten Programmiersprache untergebracht worden ist. Aus dem Artikel von Nikolaus Wirth, History and Goals of Modula-2 vom Jahre 1984 geht hervor, dass dieses Prinzip, „Information Hiding“ das allererste mal von David Parnas zur Aussprache gebracht worden ist. [1]

2 Einführung in Modula-2

Modula-2 ist eine Programmiersprache für mehrere Zwecke, aber sie ist hauptsächlich für System Entwicklungen entworfen worden. Modula-2 ist dadurch herausgewachsen da man auf der Suche nach einer allgemeinen, effizienten und implementierbaren System Programmiersprache für Mikrocomputer war. Die Vorgänger von Modula-2 sind Pascal und Modula. Wobei Modula sich nicht durchgesetzt hat. Der Nachfolger von Modula – 2 ist Oberon. Von Modula hat sich Modula-2 den Namen, das wichtige Modul Konzept und eine systematische moderne Syntax geerbt. Und das meiste vom Rest hat Modula-2 von Pascal geerbt. Das beinhaltet insbesondere die Datenstruktur, zum Beispiel Arrays, Records, abweichende Records, Sets und Pointer (Zeiger). Strukturierte Statements beinhalten die vertrauten if, case, repeat, while, for und with Statements. Ihre Syntax ist so, dass jede Struktur mit einem eindeutigen Beendigungssymbol endet.

Die Programmiersprache Modula-2 ist maschinenunabhängig [2]. Mit der Ausnahme, dass es Abgrenzungen auf Grund der Wortlänge gibt. Das wiederum scheint ein Widerspruch zur Idee von einer System-Programmierungssprache zu sein, in welcher es möglich sein muss alle Operationen auszudrücken welche dem unterliegenden System , in diesem Fall dem Computer dazu gehören. Dieses Dilemma ist mit Hilfe von dem Modul Konzept gelöst worden. Maschinen abhängige Objekte können in spezifischen Modulen eingeführt werden

und ihre Verwendung kann dabei effektiv begrenzt und isoliert werden. Insbesondere kann die Sprache eine Möglichkeit bieten, um den Regeln auszuweichen, in Falle der Kompatibilität eines Datentypen.

Hier möchte ich 3 fundamentale Unterschiede zwischen den beiden Programmiersprachen Pascal und Modula – 2 erläutern. Dies soll dazu beitragen, dass wir verstehen, welche Mängel der Sprache Pascal durch Modula – 2 aufgehoben wurden[3].

Die Unterschiede:

1. Modula – 2 baut auf die Eigenschaft des „separate compilation“ auf, wobei bei vielen Pascal Implementierungen es hinzugefügt wird.
2. Modula – 2 ersetzt das Pascal I/O (Input/Output) Routinen Read und Write, durch das Standard Modul TTIO. Dieses Modul unterstützt die Primitiven Abläufe Read und Write.
3. Ein weiterer Unterschied, bzw. eine Erweiterung ist, dass man mit Modula – 2 die Funktion eines gleichzeitigen Zugriffs hat. Diese Eigenschaft lässt zu, dass man gleichzeitige (multitasking) Programme auf einer Portablen Art und Weise schreiben kann.

Modula – 2 führt das Modul Konzept in die Welt des System Engineerings ein. Module sind die grundlegendsten Bestandteile der Sprache Modula – 2 und sie treten in verschiedenen Formaten auf. Ein Modul kann eine Gruppe von logisch verbundenen Prozeduren zusammenführen. Ein zweites Modul kann eine komplexe Datenstuktur verbergen, und ein drittes Modul kann die Fundamentalen Datentypen in einem größeren Programm definieren. Module sind entweder unabhängig oder als Submodule in ein größeres Modul eingebettet[4]. Module sind hauptsächlich separate Programmabschnitte. Typen, Konstanten, Variablen und Prozeduren, welche in einem Modul deklariert sind, sind meistens ausserhalb des Moduls nicht sichtbar. Wiederum kann man auf die Objekte welche ausserhalb des Moduls definiert sind, nicht zugreifen.

3 Syntax und Semantik

Wie in vielen anderen Programmiersprachen gibt es in Modula – 2 Bezeichner und

Schlüsselwörter, welche reservierte Wörter sind und nicht als Modul Namen,Variablen Namen oder als Konstanten verwendet werden dürfen. Hier finden sie eine Liste von diesen reservierten Wörtern:

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

Kommentare dürfen an beliebiger Stelle im Programm zwischen Symbolen stehen und werden mit folgend dargestellt (*ich bin ein Kommentar*) Kommentare dürfen sich über mehrere Zeilen erstrecken. Es ist wichtig Kommentare im Source Code einzubauen, damit man selber als Programmierer später nachvollziehen kann was man an einer bestimmten Stelle gemacht hat. Es ist aber auch eine Hilfe für andere Programmierer, die einen fremden Code lesen und verstehen wollen. Kommentare sind bei Teamarbeiten, wo mehrere Programmierer zusammenarbeiten, unbedingt notwendig.

Wie wir vorher schon erwähnt haben ist Modula-2 eine Erweiterung von Pascal. Es erweitert Pascal um das Modul Konzept. An dieser Stelle möchte ich näher in das Modul, eines der Hauptbestandteile der Sprache Modula – 2 eingehen. Module können voneinander unabhängig sein, sie können aber auch als ein Submodel in ein Hauptmodul eingebettet sein. Wir unterscheiden zwischen 3 Modulen. Dem Programm Modul, dem Definitionsmodul und dem Implementierungsmodul. Das Programm Modul ist das Hauptmodul. Das Definitionsmodul enthält alle Deklarationen. Das Implementierungsmodul unterscheidet sich von dem Programm Modul nur dadurch, dass es mit dem Schlüsselwort IMPLEMENTATION MODULE anfängt.

Hier möchte ich ein Beispiel für ein Modula – 2 Programm geben:

Beispiel für ein Modula-2 Programm, PrimZahlen:

```
MODULE PrimZahlen;
FROM InOut IMPORT WriteString, WriteInt, WriteLn;
CONST Max = 100; (* 2 < Primzahlen < 2*Max+1 *)
VAR Zahlen : ARRAY [1..Max] OF BOOLEAN;
    I,J    : INTEGER;
BEGIN
      WriteString('Sieb des Eratosthenes',0);
      WriteLn;
      FOR I:= 1 TO Max DO Zahlen[I]:= TRUE END;
      FOR I:= 3 TO (2*Max+1) DIV 3 BY 2 DO
          FOR J:= 3 TO (2*Max+1) DIV I BY 2 DO
              Zahlen[I*J DIV 2] := FALSE
          END
      END;
      J:= -1;
      FOR I:=1 TO Max DO
          IF Zahlen[I] THEN
             J:= J+1;
             WriteInt(2*I+1,6);
             IF J MOD 10 = 9 THEN WriteLn END
          END
      END; WriteLn
END PrimZahlen.
```

Das Programm fängt mit einem Modulkopf an. Es wird durch `MODULE` gefolgt vom Modulnamen `Primzahlen`, angegeben. Durch die IMPORT – Anweisung, werden Prozeduren WriteString, WriteInt und WriteLn aus einem bereits existierenden Modul InOut in das Programm eingelesen. Dies ist eine Eigenschaft, die Pascal nicht hatte. Und wo es Schwierigkeiten und Mühsamkeiten gegeben hat. Danach folgt die Deklaration von Konstanen Max, Variablen und Integer. Danach folgend die Anweisungen. Jede Variable, die im Anweisungsteil verwendet wird oder aufgerufen wird, muss vorher deklariert worden sein. Es gibt die Deklaration von Typen, Konstantendeklaration, Variablendeklaration und Prozedurdeklaration. Das Modul PrimZahlen wird durch das Beendigungssymbol END

Primzahlen. beendet. Auch jede IF oder FOR Anweisung wird mit END; beendet. Hier ist wichtig zu erwähnen, dass Modula – 2 eine case – senstive Sprache ist. Das bedeutet, sie unterscheidet, zwischen Gross- und Kleinschreibung. Ein Modul endet mit END Modulname. und eine Anweisung endet mit END; Hier muss man auf den Punkt und den Semikolon achten. Da Module separate eigenständige Programmabschnitte sind, sind sie auch einzeln Kompilierbar. Dies war bei Pascal nicht möglich.

Auch in Modula-2 gibt es Wertzuweisungen, IF-, CASE-, WHILE-, REPEAT-, FOR-, WITH- und RETURN-Anweisungen und den Prozeduraufruf. Den Prozeduraufruf kennen so manche auch von Pascal. Der Prozeduraufruf aktiviert eine Prozedur. Eine Prozedur kann auch rekursiv aufgerufen werden.

Bei diesem Punkt möchte ich noch ein Beispiel geben, das Beispiel über eine Stack Operation:

```
DEFINITION MODULE Stack;
        TYPE
                Stack = RECORD
                top: INTEGER;
                value: ARRAY [0..999] OF INTEGER;
        END;
        PROCEDURE Clear (VAR s: Stack);
        PROCEDURE Push (VAR s: Stack; i: INTEGER);
        PROCEDURE Pop (VAR s: Stack; VAR i: INTEGER);
        END Stack.
        IMPLEMENTATION MODULE Stack;
                PROCEDURE Clear (VAR s: Stack);
                BEGIN
                        s.top := 0;

                END Clear;
                PROCEDURE Push (VAR s: Stack; i: INTEGER);
                BEGIN
                        IF s.top <= 999 THEN
                        s.value[s.top] := i; INC(s.top);
                        END;
                END Push;
                PROCEDURE Pop (VAR s: Stack; VAR i: INTEGER);
                BEGIN
                        IF s.top > 0 THEN
                        DEC(s.top);
                        i := s.value[s.top];
                        END;
                END Pop;
                END Stack.
```

Hier sehen wir sehr deutlich das Definitionsmodul (`DEFINITION MODULE Stack;`) und das Implementierungsmodul (`IMPLEMENTATION MODULE Stack;`). Wobei hier Stack der Name des Moduls ist. Im Definitionsmodul sehen wir die Typdeklaration. Es wird ebenfalls mit einem END; beendet. Danach folgen drei Prozeduren (`PROCEDURE Clear, PROCEDURE Push und PROCEDURE Pop`). Der Prozeduraufruf folgt dann im Implementierungsmodul, wo dann die Prozeduren aktiviert werden. Jede Prozedur endet folgend, END Prozedurname; In unser Beispiel wäre die Beendigung der Prozedur `PROCEDURE Clear,` durch `END Clear;` erfolgen. Hier wird das Modul mit END Stack. beendet.

4 Vor – und Nachteile

Wie in jeder Sprache gibt es Vor – und Nachteile. Die Vorteile von Modula – 2 sind, dass die Sprache leicht zu lernen, zu lesen und zu verstehen ist. Der Code ist übersichtlicher als in Pascal. Die Module helfen dem Programmierer den Source Code zu Verfeinern und zu Zerlegen. Dadurch kommt man den Problemen näher. Ein weiterer Vorteil von Modula – 2 ist, dass es eine umfangreiche Laufzeitbibliothek hat. Was bei Pascal nicht der Fall ist. Des weiteren hat Modula – 2 ein ausgeprägtes File I/O Konzept, was zum Beispiel bei Pascal fehlt. Ein weiterer Vorteil ist, dass in Modula – 2 die Möglichkeit entstanden ist die Konstante andere Konstanten oder auch Ausdrücke enthalten kann.

Zum Beispiel:

```
Sekunde = 1;
Minute = 60 * Sekunde;
```

Als jedoch Nachteil von Modula – 2, ein womöglich wichtiger Grund warum Modula – 2 sich am Markt nicht durchgesetzt hat ist, dass die Modul-Library nicht genormt ist.

5 Zusammenfassung

Modula – 2 hat eine wichtige Bedeutung in der Welt der Programmiersprachen, weil

es das Modul Konzept eingeführt hat. Auch das Information Hiding Prinzip wurde erstmals mit Modula – 2, mit einer klar deklarierten eigenständigen Programmiersprache umgesetzt. Diese Programmiersprache eignet sich, wie sein Vorgänger, als Unterrichtswerkzeug, um den Schülern das Konzept der Programmierung beizubringen. Es ist sicher ein guter Einstieg für Programmier-Beginner. Der Nachfolger von Modula – 2 ist Oberon, ebenfalls von Nikolaus Wirth entwickelte Objektorientierte Programmiersprache. Modula – 2 eignet sich für System Entwicklungen und grössere Projekte.

6 Literaturliste

[1] N.Wirth , History and Goals of Modula-2, Byte , 1984

[2] N.Wirth, Modula-2, ETH Zürich, 1980

[3] R.J.Paul, An Introduction to Modula-2, Byte, 1984

[4] J.Gutknecht , Tutorial on Modula-2, Byte, 1984

[5] J.Gutknecht, Programming Languages and System Archtiectures, Springer, 1994

[6] http://www.modula2.org