Einführung in die Programmiersprache Extended Pascal

HINTERGRUND

Ähnliche, aber nicht identische Standards für Pascal wurden 1983 durch ISO (ursprünglich ein BSI Standard) und von ANSI veröffentlicht. Während die Definition, die sie enthielten sehr exakt war, versuchte es nicht, viel zur Wirth `s Vorlage Spezifikation hinzuzufügen. Eine Konsequenz war, dass, während viele Implementationen die Gemeinsprache liefern, sie auch ihre eigenen Verlängerungen planten, um Benutzer mit Eigenschaften zu versehen, die gefunden wurden, um auch der Nachfrage zu entsprechen.

Nach der Publikation des ersten Standards, arbeitete der ANSI/IEEE Gemeinschaftsprojektausschuss, der fortgesetzt wurde, um Steigerungen zu entwickeln, von 1984 an mit der ISO Gruppe zusammen, um die Definition einer ausgedehnten Pascalsprache *(Extended Pascal)* zu produzieren, die mit den früheren Versionen kompatibel sein sollte. *Extended Pascalstandards*, welche im technischen Inhalt identisch waren, wurden 1990 und 1991 veröffentlicht.

Nachdem sie mit dem Standard fertig waren, fuhren die technischen Ausschüsse fort, neue Eigenschaften in den Bereichen der orientierten Programmierung des Gegenstandes und der Ausnahmebehandlung zu entwickeln.

Extended Pascal ist ein wenig größer als der frühere Standard (auf den im Allgemeinen als "Classic Pascal" bezogen wird) und eine Einleitung dieser Art kann nur eine allgemeine Umfrage liefern. Wie auch immer ist die erste komplette Implementierung für die Mikrocomputer produziert worden, die sich auf DOS laufen lassen, also ist es angemessen festzustellen, dass das ursprüngliche Ziel von Professor Wirth für Pascal, dass es für leistungsfähige Implementierung auf zur Zeit vorhandenen Rechnern verwendbar sein sollte, immer im Sinn behalten werden sollte.

ALLGEMEINE VERBESSERUNGEN (General Enhancemets)

Extended Pascal enthält einige neue wichtige Eigenschaften, aber, bevor diese betrachtet werden, ist es nützlich, eine Übersicht mit einigen Verbesserungen zu beginnen, die es ermöglichen, direkt vom klassischen Pascal zu übernehmen und einfacher zu verwenden.

KONSTANTENAUSDRÜCKE (Constant Expressions) können, wo klassisches Pascal Konstanten erfordert, z.B. um sie in den Erklärungen zu verwenden. Diese Konstantenausdrücke können die meisten vorbestimmten Funktionen, sowie Operatoren eingesetzt werden.

ENTSPANNENDE EINRICHTUNG (relaxed ordering).
Die Erklärungen und die Definitionen (CONST, Art, var usw..) können wiederholt werden und können in jedem möglichem Auftrag erscheinen, vorausgesetzt dass es keine weiteren Verbindungen gibt.

FUNKTIONEN (functions) können Resultate irgendeiner bestimmbaren Art, einschließlich Reihen (*arrays*) und Aufzeichnung (*record*) zurückbringen, und auch, wenn sie passend von der Eigenschaft (*type*) sind, Prozesse von dereference(^), von Index-Bewegung und von Feldvorwählern. Der Ausgangsvariable (*result variable*) kann ein Name gegeben werden (unterschiedlich zu dem Funktionsnamen), auf den im Kontext bezogen werden kann, der eine Variable erfordert, ohne einen rekursiven Anruf der Funktion zu verursachen.

SUBRANGE GRENZEN (subrange bounds) können Konstantenausdrücke, in Übereinstimmung mit der weitgehenden Entspannung, die oben beschrieben wird, sein und solche *subranges* definieren herkömmliche Grenzen des *Static types*. Die Grenzen können auch die allgemeinen Ausdrücke sein und beziehen Laufzeitvariablen mit ein, und auch *nonstatic types*. Auf dieses Thema wird separat später noch eingegangen.

FALL-VERBESSERUNGEN (Case enhancements)
In beiden Varianterekorderklärungen und in den CASE…OF (Fall)- Aussagen sind Anwendungen in den Konstantenlisten erlaubt, und eine OTHERWISE (Anders) Klausel kann eingeführt werden, um die Liste durchzuführen.

```
CASE ch OF
  '0'..'9'            digit;
  'A','E','I','O','U': vowel;
   OTHERWISE          other;
END (case);
```

SHORT-CIRCUIT OPERATOREN (short circiut operators)
Varianten der Booleschen Operatoren UND und ODER werden zur Verfügung gestellt, die garantieren, dass ein Ausdruck nicht weiter ausgewertet wird, als notwendig, um das Ergebnis festzustellen. Die neuen Varianten werden AND _ THEN und OR_ELSE genannt.

NONDECIMAL ZAHLEN (Non decimal numbers) können in Quellenprogrammen mit der Darstellung *base#number*, zum Beispiel 16#FF oder dem 8#377 (hexadezimales FF=octal377=decimal255) verwendet werden.

CHARACTERISTICS (Eigenschaften)
So wie *maxint* eine Implementierung-spezifische Information über die Ganzzahlart angibt, gibt es ein vorbestimmtes konstantes *maxchar* (der Buchstabe mit größtem Ordnungswert), und die *maxreal, minreal* und *epsreal* Konstanten, die die Eigenschaften der realen Art angeben.

NUMERISCHER EINGANG (numeric input) nimmt von den textfiles die Darstellung an, die im Daten-Austausch Standard (ISO 6093) niedergelegt wird, der ein Dezimalkomma in führender oder schleppender Position ermöglicht.

NULL FIELDWIDTH(zero fieldwidth)
Die wahlweise freigestellter *fieldwidth* Parameter im textfile Ausgang kann den Wert null einnehmen.

UMGEKEHRTES ORD.
Eine Verbesserung zum *succ* und *pred* Funktionen lässt einen wahlweise freigestellten zweiten Parameter der Ganzzahl annehmen. Dieses ist vielseitiger begabt als ein einfaches Gegenteil von *ord*, aber gibt unter anderem *succ*, die Fähigkeit eine herkömmliche Tag-von-Woche Aufzählung auszugeben. Z.B: succ(Monday, 3) gibt Donnerstag aus.

UNTERSTREICHEN (underscores) sind als bedeutende Buchstaben innerhalb der Bezeichner erlaubt, jedoch nicht in führenden oder schleppenden Positionen.

AUSGANGSZUSTÄNDE (Initial States)
Im klassischen Pascal werden alle Variablen in nicht definiertem Zustand erzeugt (das heißt, einen undefinierte Eigenschaft beinhaltend). In *Extended Pascal* kann ein Ausgangszustand definiert werden, der automatisch zu einer Variablen gegeben wird, wenn er erzeugt wird. Auf dem äußeren Niveau kann eine Implementierung den Ausgangszustand

einstellen, aber der Service trifft auch an den lokalen Variablen von Verfahren und auf die Variablen zu, die auf dem Haufen verursacht werden. Außerdem kann die Definition einer Art einen Ausgangszustand tragen, welche auf jede Variable derselben Art übertragen wird, es sei denn die Variablen- Erklärung selbst überläuft sie.

REIHE UND SATZ-ERBAUER (Array and record constructors)

Die Satzerbauer des klassischen Pascal bilden die Grundlage der Erbauer, um andere strukturierte Werte zu definieren. Wenn alle Bestandteile konstant sind, kann ein konstruierter Wert als Konstante genannt werden oder verwendet werden, um Ausgangszustände zu definieren. Innerhalb der Aussagen kann der Erbauer Laufzeitwerte einschließen.

Ein Reihe Erbauer kann spezifische Indizes verzeichnen, mit den Werten sollen jene Elemente gegeben werden und/oder können eine Rückstellung zu allen möglichen gegebenen Elementen zur Verfügung stellen, die nicht einzeln verzeichnet wurden. Ein Aufzeichnungs-Erbauer benennt Aufzeichnungsfelder zusammen mit den Werten, die gegeben werden sollen; die ganze Aufzeichnung muss definiert werden.

MODULE (Modules)

Klassisches Pascal schließt keinen Service für unterschiedliches Kompilieren der Teile eines Programms ein. Außer dem Begrenzen des Bereichs der Programme, die auf kleinen Maschinen produziert werden können, hat dieses den großen Nachteil, dass es keine Standardform für die Vorbereitung von vorkompilierten Bibliotheken gibt. Fast jede Implementierung von Pascal stellte eine Steigerung irgendeiner Art vor, um diese Beschränkung zu überwinden, und es wurde als eine der wichtigsten Aufgaben von *Extended Pascal* gesehen, eine Form des unterschiedlichen Kompilierens zu definieren, die nicht an Sicherheit der Art einbüßen würde.

Außer dem Hauptprogramm können *Extended Pascal*-Programme die Bestandteile einschließen, die als Module bekannt sind. Ein Modul kann Konstanten, Arten, Variablen, Verfahren und Funktionen durch genannte Schnittstellen exportieren, und diese Schnittstellen können durch andere Module oder durch das Hauptprogramm importiert werden. Normalerweise eine Schnittstelle wird komplett importiert, mit den Namen aller seiner zugänglichen Bestandteile, aber es gibt einige Optionen, wo man auf Schwierigkeiten treffen kann, welche beim Anwenden entstehen können, wenn man von Modulen importiert, die nicht dafür gemacht wurden, miteinander zu arbeiten. Anstatt die ganze Schnittstelle zu importieren, sollte man vorgewählte Einzelteile wählen; und Bestandteile können beim Import umbenannt werden.

Ein Modul hat zwei Teile; eine Überschrift und ein Block. Die Modulüberschrift enthält Erklärungen und Definitionen aller möglicher Einzelteile, die exportiert werden sollen, insbesondere die Überschrift (aber nicht mehr) von Prozeduren und Funktionen. Der Block schließt die Definitionen aller möglicher exportierten Prozeduren oder Funktionen, zusammen mit Einzelteilen ein, die außerhalb des Moduls nicht bekannt sein müssen. Es kann auch Initialisierungs- und Vollendungscode sein.

Die Überschrift und der Block können kombiniert werden oder sich trennen. Wenn sie unterschiedlich sind, entsteht die Möglichkeit von den alternativen Implementierungen der gleichen Überschrift (mit und ohne Diagnose-Code, z.B.) oder der Implementierung, die in irgendeiner anderer Sprache wie Versammlungsteilnehmer kodiert wird.

Eine Modulüberschrift kann aus einem anderen Modul importiert werden und kann diese importierten Einzelteile wieder exportieren und erlauben, dass zum Beispiel zusammengesetzte Bibliothekschnittstellen konstruiert werden. Ein Modulblock kann

Schnittstellen von anderen Modulen unabhängig importieren. Der Export und der Import einer Schnittstelle stellen auf, was als "supplying" (beweglich) Verhältnis bekannt ist, in dem das exportierende Modul das importierende Modul oder das Programm liefert. Das Versorgungsmaterial-Netz setzt einige Begrenzungen auf die Reihenfolge, in der Module kompiliert werden können; insbesondere darf es keine Schleifen (die enthalten würden, dass ein Modul indirekt versuchen würde, sich selbst zu bewegen).Modulinitialisierungscode wird vor irgendeinem Bestandteil, den er liefert, durchgeführt.
Modularer Aufbau ist normalerweise für die größeren Programme angebracht, und kleine Beispiele erscheinen unvermeidlich trivial. Wie auch immer, die drei angeführten Module, sollen eine Anzahl der Möglichkeiten demonstrieren.

ZEICHENKETTEN (Strings)

Im klassischen Pascal ist der einzige Zeichenkette-Service mit gepackten Arrays verbunden. Dieses ist ein anderer Bereich, in dem eine Vielzahl der lokalen Steigerungen entstanden ist. *Extended Pascal* schließt die Bestimmung für dynamische Zeichenkettearten ein und vereinheitlicht sie mit klassischen Pascalzeichenketten und mit Buchstaben.

Zeichenkettevariablen werden mit einer maximalen Kapazität, zum Beispiel erklärt:

```
VAR  s1,s2: string(20);
   fname: PACKED ARRAY [1..20] OF char;
```

Zeichenkettewerte haben eine *length*(Anzahl der Buchstaben). Eine dynamische Zeichenkettevariable wie s1 kann einen Wert irgendeiner Länge von null bis zu seiner Kapazität beinhalten, und die Objektcode-Unterhaltschiene des Stromes ist der Kapazität gleich; wenn ein kürzerer Wert *fname* zugewiesen wird, wird er so lange aufgefüllt bis die Kapazität erlangt ist. Eine Variable der Art *char* hat eine Kapazität von 1. Variablen dieser drei Arten, zusammen mit Zeichenkettendruckfehlern und Zeichenkonstanten, produzieren allgemeine Zeichenkettenwerte. Zusätzlich können einzelne Buchstaben oder Teilketten der Zeichenkettevariablen durch das Registrieren, zum Beispiel s1 [i] oder fname [1..8] bezogen werden.

Wenn eine Variable n den Wert 10 hat, hat eine Zeichenkette, die als *string (n)* erklärt wird, die gleiche Art wie eine, die als string(10) zwecks der Kompatibilität erklärt wird (obwohl die Art Überprüfung nicht bis zur Laufzeit durchgeführt werden kann). Wie wir im folgenden Abschnitt sehen werden, sind diese Richtlinien und die anpassungsfähigen Formalparameter und die Zeigerarten bestimmte Fälle vom Service, der mit allen schematischen Arten vorhanden ist, und aus der Zeichenkette entstehen, die formal definiert wurde, um ein vordefiniertes "Schema" mit zusätzlichen speziellen Eigenschaften zu sein.

NICHTSTATISCHE ARTEN (non static types)

Es ist eine Eigenschaft fast aller Versionen von Pascal gewesen, dass die Daten statisch sind, d.h. schließlich nur beim Kompilieren definiert werden. Die ISO Version des klassischen Pascal schloss eine Zusatzeinrichtung ein, die *conformant array parameters*, eine fachkundige Parameterform für welche Aktualarrays von unterschiedlicher Größe in den unterschiedlichen Anrufen das gleiche Verfahren geliefert werden können. Diese Eigenschaft ist in einer Anzahl von Implementierungen eingeschlossen worden und liefert ein Maß an Flexibilität, die z.B. mathematischen Verfahren entspricht, die Reihen manipulieren. Alles, das angefordert wird, ist dass dieser Aktualparameter sich dem formalen anpasst im Sinne von derselben Anzahl an Maß und abschließender Elementart.
Im Kontext des klassischen Pascal, geben *conformant array parameters* einen Grad an Flexibilität, wenn gebrauchsfertige Verfahren in einen Programm in der Quellform eingeschlossen werden, jedoch müssen alle Aktualparameter ausschließlich statisch sein, mit zur Laufzeit definierten Größen. *conformant array parameters* sind eine

Zusatzeinrichtung von *Extended Pascal*, aber es gibt zusätzlich eine weiterreichende Vielzahl der nicht statischen Arten, die auf Schemata basieren.

Ein Schema ist eine Schablone, die eine Familie der in Verbindung stehenden Arten beschreibt, aus denen einzelne Arten durch Ersetzen entweder in der Kompilierzeit- oder Laufzeit Werte produziert werden können, um gewöhnlich *subrange* Grenzen zu definieren oder Rekordvarianten vorzuwählen. Dieses "schematischen" Arten können in fast immer wie herkömmliche statische Arten benutzt werden; sie können in der Erklärung von Variablen, von Rekordfeldern und von Formalparametern benutzt werden; sie können als Gebiet Arten der Zeiger benutzt werden, und durch Funktionen zurückgebracht werden. Es wurde früher beobachtet, dass *subranges* ihre Grenzen haben können, die in der Laufzeit festgestellt werden; solche *subranges* sind einzelnen *schematictypes* ohne dem Nutzen einer Familienverbindung.

Formalparameter können mit dem ursprünglichen Schemanamen erklärt werden und werden sich dem Aktualparameter an jedem Anruf, wie vorher schon beschrieben für bestimmte Fälle von Zeichenketten. In dieser Hinsicht sind sie *conformant array parameters* ähnlich, aber erfordern, dass jeder tatsächlich von einer Art ist, die aus dem gleichen Schema produziert wird. Ein Zeiger kann auch erklärt werden, so dass er den Schemanamen als seine Gebiet Art hat, und eine zusätzliche Form der Prozedur *new* wird zur Verfügung gestellt, das tatsächliche Werte beinhaltet, um eine Art von Schema vorzuwählen.

Ein Schema kann eine Familie von Aufzeichnungsarten definieren, in welchem eine Variante durch einen Parameter des Schemas vorgewählt wird. Die Auswahl kann entschieden werden zur Laufzeit, und anders als die Form von *new* übernommen vom klassischen Pascal, erfordert sie keinen konstanten Auswähler. Wie mit allen schematischen Arten, können solche Aufzeichnungen lokale Variablen, Parameter, Felder von anderen Aufzeichnungen sein und so weiter. Die Wahl der Variante produziert eine spezifische Art, die nicht nachher geändert werden kann; aber wie mit jedem möglichem Schema kann ein Parameter mit dem Schemanamen erklärt werden, der als Aktualparameter irgendwie die produzierten Arten annimmt. Der Gebrauch von verschiedenen Aufzeichnungen wird sicherer und flexibler durch diese Vorbereitungen gebildet.

ANFRAGE DER ART (Type inquiry)

Diese Eigenschaft ist vom Gebrauch hauptsächlich in Verbindung mit Schemaarten erlaubt, dass zum Beispiel eine lokale Arbeit Variable mit der gleichen Art wie ein Aktualparameter erklärt wird, wenn diese Art nicht bis Laufzeit bekannt und von Anruf zu Anruf sich unterscheiden kann.

MATHEMATISCHE STEIGERUNGEN (Mathematical extensions)

Eine komplizierte Datenart wird zur Verfügung gestellt. Sie ist absichtlich undurchlässig, um Implementierungen zu ermöglichen, die passendste Darstellung zu wählen; es gibt die Funktionen, um die Real- und Imaginärteile (eine kartesische Ansicht), die Größe und das Argument (eine polare Ansicht) zu erhalten, und eines komplizierten Wertes von jedem Paar Eingängen auch zu konstruieren. Die mathematischen Operatoren und die Funktionen des klassischen Pascal können komplizierte Argumente auch aufnehmen und komplizierte Resultate zurückbringen.

DATEI STEIGERUNGEN (File extensions)

Extended Pascal stellt eine Methode des Bindens einer Variablen innerhalb des Programms zu einem externen Wesen zur Verfügung; ein allgemeines Beispiel ist die Bindung einer Dateivariablen zu einer Betriebssystem Datei. Es gibt eine vordefinierte Satzart, die *BindingType* genannt wird, und die verbindliche Informationen enthält; Die Prozeduren bind

und *unbind* führen die Tätigkeiten durch, und eine Funktion *binding* bringt die gegenwärtige Lage einer Variable zurück. Das Dokumentbinden kann durch eine Reihenfolge von Operationen durchgeführt werden, die verhältnismäßig unabhängig von der Umgebung sind; einige andere Bindungen (wie zu einem Bildschirmabbild oder einer Uhr) können in den spezifischen Implementierungen vorhanden sein.

Klassisches Pascal stellt nur aufeinander folgende Dateiverarbeitung zur Verfügung. *Extended Pascal* addiert die Fähigkeit, um eine aufeinander folgende Datei zu verlängern und erlaubt auch, dass Dateivariablen mit einer Indexart erklärt werden. Solche Variablen können direkten Zugriff zu den einzelnen Dateielementen, durch Spezifizieren eines Indexwertes zur Verfügung gestellt werden. Direktzugriffsdateien erlauben das Aktualisieren sowie das Lesen und Schreiben.

SATZ STEIGERUNGEN (set extensions)

Ein neuer Operator >< wird definiert, der die symmetrische Differenz von zwei gesetzten Werten nimmt; es gibt eine neue vorbestimmte Funktion *card*, die die Kardinalität eines Satzes zurückgibt (die Anzahl der anwesenden Mitglieder);

DATUM UND UHRZEIT (date and time)

Ein vordefinierte Satzart *TimeStamp* wird zur Verfügung gestellt, die Felder für Jahr, Monat, Tag, Stunde, Minute und Sekunde enthält. (es wird beabsichtigt, dass eine Implementierung weitere Details wie Millisekunde oder Zeitzone hinzufügen konnte, die durchsichtig durch die vorbestimmten Programme verarbeitet werden würde) Eine Prozedur *GetTimeStamp* stellt die gegenwärtigen Werte in einer TimeStamp- Aufzeichnung ein; Funktionen *Date* und *Time* nehmen eine solche Aufzeichnung wie ein Parameter auf und geben Zeichenketten im Anzeigeformat zurück.