Quaestiones Informaticae

An official publication of the Computer Society of South Africa
'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika

Editors: Dr. D. S. Henderson,
Vice Chancellor, Rhodes University, Grahamstown, 6140, South Africa.
Prof. M. H. Williams,
Department of Computer Science and Applied Maths,
Rhodes University, Grahamstown, 6140, South Africa.

Editorial Advisory Board

PROFESSOR D. W. BARRON
Department of Mathematics
The University
Southampton S09 5NH
England

PROFESSOR K. GREGGOR
Computer Centre
University of Port Elizabeth
Port Elizabeth 6001
South Africa

PROFESSOR K. MACGREGOR
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch 7700
South Africa

PROFESSOR G. R. JOUBERT
Department of Computer Science
University of Natal
King George V Avenue
Durban 4001
South Africa

MR. P. P. ROETS
NRIMS
CSIR
P.O. Box 395
PRETORIA 0001
South Africa

PROFESSOR S.H. VON SOLMS
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg 2001
South Africa

PROFESSOR G. WIECHERS
Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

MR. P. C. PIROW
Graduate School of Business Administration,
University of the Witwatersrand
P.O. Box 31170
Braamfontein 2017
South Africa

Subscriptions

Annual subscriptions are as follows:

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>US</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals</td>
<td>R2</td>
<td>$3</td>
<td>£1.50</td>
</tr>
<tr>
<td>Institutions</td>
<td>R4</td>
<td>$6</td>
<td>£3.00</td>
</tr>
</tbody>
</table>

Quaestiones Informaticae is prepared for publication by SYSTEMS PUBLISHERS (PTY) LTD for the Computer Society of South Africa.
Design Principles of the Language BPL

M.H. Williams

Dept. of Computer Science and Applied Maths, Rhodes University, Grahamstown, South Africa.

Abstract

BASIC and Pascal are two rather different programming languages, both of which have proved to be very popular. A new language BPL has been designed as an interactive programming language based on these two languages, combining advantages of each. This paper discusses some of the criteria used in the design of the language.

1. Introduction

The programming language BASIC was originally created by J.G. Kemeny and T.E. Kurtz [4] of Dartmouth College, U.S.A., as a language for teaching programming in an educational environment [2, 4]. Its simplicity has led to its popularity and currently it is widely used for teaching purposes. In addition it is the high-level language which is most common on microcomputers and mini-computers today.

Pascal was also created as a language for teaching programming [3, 7] and while it is very different from BASIC, it too has proved to be very popular. The main reasons for this are undoubtedly the wide range of data types and the neat control structures available in the language [5].

Both languages do, however, have certain disadvantages and it was for this reason that the language BPL has been proposed — once again as teaching language. BPL [6] aims at combining the simplicity of BASIC with the wide range of data types and neat control structures of Pascal to produce an interactive language which on the one hand is easy to teach and learn while on the other offers a powerful and flexible tool for solving problems. BPL is currently being implemented in Pascal on an ICL 1900 computer. The purpose of this paper is to examine some of the design decisions embodied in BPL.

2. General Principles

The chief advantages of the language BASIC are considered to be the following:

(a) The concept of combining programming language and system command language into one system is a very useful one especially for beginners who are learning to use an interactive system. It is also very useful in simple microprocessor systems where savings can be effected by combining these two. The particular set of commands available in most BASIC systems is based on the commands NEW, OLD, SAVE, LIST, RESEQUENCE, TRACE, BREAK, etc., as proposed by Kemeny and Kurtz and extended by Bull [1]. These are simple and easy to use, and are generally ideal for most purposes.

(b) Input-output (via the console or disc files) is simple for the beginner but is also sufficiently flexible to handle most advanced applications too.

(c) The notion of only two data types (reals and strings), the association of a data type with the form of the identifier and the automatic allocation of variables is considered to be an advantage by many users.

(d) The modus operandi of the system, viz. syntax checking as each statement is entered into the system and the ability to correct it immediately if it is incorrect, the ability to stop a program and access its variables and then continue execution, and so on, is very useful from the programmer’s point of view.

(e) String variables are simple to use and yet the operations available are powerful.

On the other hand Pascal has the following important advantages:

(a) The neat control structures (with the exception of GOTO) which are available in the language make it easy for the user to write neat and well-structured programs.

(b) The wide range of data types available assists the user in producing more readable programs. Indeed everything that one can do with defined scalar variables, pointer variables, or even record variables can be achieved in other ways in other languages — however, the result is not necessarily as readable as it is in the case of Pascal.

(c) The notion of having to declare any variable before it is used, provides a useful check on the programmer’s intent.

(d) The length of an identifier in Pascal is much more convenient than that in BASIC and also contributes towards producing more readable programs.

The aim of BPL is to combine the advantages of both of these systems. BPL was designed as two-tier language with the following levels:

(a) The lower level is a subset of the complete language intended for beginners. It is very similar to BASIC and has two data types — real and variable length strings. All variables are allocated automatically although all arrays must be declared.
(b) The higher level is the complete language, in which a much wider range of data types is available, viz. real, integer, Boolean, defined scalar, subrange, record, pointer, set, fixed-length string or variable-length string. One may also have arrays of any of these types. The set of statements available at the higher level is almost identical to that available at the lower level, with the exception of a few additional statements to cater for the extra data types. The set includes many of the standard statements from BASIC with the exception of the control statement which are replaced by structured forms.

3. Some Aspects of the Language

3.1 Identifiers

As in Pascal identifiers may be of any length although only the first \( n \) characters are significant (where \( n \) is an implementation dependent limit value \( \geq 8 \) in Standard BPL). In a similar fashion to BASIC an identifier may be one of two types:

(i) A plain identifier — which consists of a sequence of letters and digits beginning with a letter.

(ii) A string identifier — which consists of a sequence of letters and digits beginning with a letter, and followed by a $ sign. String identifiers are used for variables, arrays or functions of type string while plain identifiers are used in all other instances. The idea in BASIC of having a single variable and an array bearing the same name has always caused confusion amongst beginners and thus in BPL variable, array and procedure names must be unique. One consequence of this requirement is that one need not distinguish between READ, PRINT, etc., and MATREAD, MATPRINT, etc. If any array name without subscripts appears in the destination list of a READ statement, values are to be read into the whole array; likewise for a PRINT statement the whole array is to be printed.

3.2 Control Statements

The statement numbers in BASIC serve a dual purpose — they act as line numbers for the purpose of editing programs and they serve as destinations for GOTO statements. This aspect of BASIC has been criticised since it is contended that the presence of labels encourages the use of GOTO statements which in the long run leads to poorly structured programs. To avoid this it has been decided to omit the GOTO statement altogether from BPL's repertoire of statements. Structured forms of IF-THEN and IF-THEN-ELSE statements are already available in many BASIC systems and these have been adopted in BPL. The FOR-NEXT loop in BPL is a simple extension of the BASIC loop (this is in turn more powerful than the for loop in Pascal which does not permit the controlled variable to be of type real nor allows for any increment other than \(+1\) or \(-1\)). Both WHILE and REPEAT loops similar to those available in some BASIC systems have also been included.

The multiway branch is an ON statement with a form similar to the case statement in Pascal except that it permits a default case, e.g.

```
100 ON I
110 CASE 1: LET A=1
120 B=2
130 CASE 2: READ A,B
140 I=0
150 DEFAULT: A=0
160 ENDON I
```

To compensate for the lack of a GOTO statement, an EXIT statement is included to enable one to exit from within a loop or a set of nested loops. This has the form

```
EXIT loop variable — exits from the loop with this loop variable
or EXIT — exits from the innermost loop without a loop variable
```

3.3 Procedures and Functions

Another obvious feature which aids good programming and which is lacking in most BASIC system, is a good procedure/function mechanism. This is essential for top-down programming and for increasing the readability of programs. In BPL a procedure or function is introduced by a procedure declaration statement of form:

```
PROCEDURE<proc id>|(<formal par>)
|,|,<formal par>,|0|0|0
```

or

```
FUNCTION<function id>|(,<formal par>)
|,|,<formal par>,|0|0|0
```

This is followed by the statements comprising the procedure body and terminated by ENDPROC.

Unlike BASIC the statement numbers of statements within a procedure are completely independent of any statement numbers occurring outside that procedure. This facilitates the inclusion of procedures into a program, separate compilation of procedures onto a library, etc. On the other hand it does complicate system commands such as TRACE, BREAK, LIST, etc. Procedures cannot be nested as in Pascal, but they may have their own local variables and they may call each other recursively. Parameters are called by reference as in BASIC — there is no call by value as there is in Pascal.

3.4 Declarations

The approach towards declarations adopted in BPL is similar to that taken in Pascal in that types, global variables and arrays are declared at the beginning of the program. This is followed by any procedures and functions and the main body of the program. As will be noticed from the discussion on the advantages of BASIC and PASCAL, it is regarded by some to be an advantage to have automatically allocated variables, while others prefer that all variables should be declared before use. BPL attempts to cater for both points of view by providing three modes of operation:

(a) Declaration mode, in which every identifier must be declared before it is used,

(b) Automatic allocation with cross-reference, in which any identifier encountered which has not been declared, will be assumed to be either a procedure or a variable or function of type variable-length string (if the identifier is a string identifier) or real (if it is not). At the end of each segment (procedure, function or main body) a list of all automatically allocated variables is given. No arrays are allocated automatically.

(c) Automatic allocation without cross-reference, which is similar to (b) except that cross-reference tables are not produced.

In BPL two mechanisms are provided for array declaration, viz.

(a) An array may be declared using an array type, e.g.

```
10 TYPE ARRTYPE = ARRAY[1..5]OF REAL
20 VAR X:ARRTYPE
```

This declares static arrays as in Pascal.

(b) The DIM statement may be used, e.g.

```
50 DIM A(5),B(N)
```

This may be used to declare dynamic arrays. Once an array has
been set up in a segment by a DIM statement, its size and bounds are fixed until execution of that segment is complete and control is returned to the calling segment. The DIM statement is an extension of the BASIC form of DIM.

3.5 Input-output
There are five input-output statements based on those available in BASIC. These are: READ, PRINT, DATA, FILE and RESET. In addition there are three functions: TAB, NL and EOF. The statement READ can have various forms such as

10 READ A,B,C — read 3 values from
DATA statement into A,B,C
10 READ#1: A,B,C — read 3 values from channel 1 into A,B,C
10 READ#1: USING BINARY:— read from channel 1 into A,B,C using binary format

The items in the destination list may be simple variables, subscripted variables, fields, records or arrays of any type except pointer or set. Output is achieved by means of the PRINT statement which also has a format similar to that in BASIC, e.g.

10 PRINT "RESULT=":A+B,PAY
20 PRINT#1:MATRlX
30 PRINT#1:USING BINARY:EMPLREC(I)

If input/output is not associated with the console, a FILE statement must be used to assign a channel number to the particular file or device required. This has format

FILE#<numeric exp>:<string exp>;<mode>]

where <string exp> reduces either to a device name (e.g. "CR", "LP", etc.) or a file name.

3.6. Expressions
The way in which string manipulation is handled in BASIC is both simple and powerful. String variables and constants may be concatenated (using the operator "+") substrings may be accessed or assigned to (e.g. A$[1,3]) strings or substrings compared lexicographically, etc. In BPL the same facilities are provided but for two types of string viz. (a) fixed-length strings of predetermined length, declared as follows:

10 VAR AS,BS:STRING(8)

(b) variable-length strings with specified maximum length, e.g.

20 VAR CS,DS:STRING(<=8)

If a string variable is not declared, it is assumed to be a variable-length string with maximum length equal to a default value. Arithmetic expressions are similar to those in BASIC and include the operations +, −, *, / and exponentiation which is denoted by ** to avoid confusion with the pointer operation !. The absence of an operator for exponentiation is one of the criticisms levelled against Pascal.

Boolean expressions are much the same as in Pascal or BASIC. The order of precedence of operators in BPL is as follows:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊕</td>
<td>1</td>
</tr>
<tr>
<td>−</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
</tr>
<tr>
<td>/</td>
<td>4</td>
</tr>
<tr>
<td>**</td>
<td>5</td>
</tr>
<tr>
<td>&gt; &gt; &gt;</td>
<td>6</td>
</tr>
<tr>
<td>&gt; &gt; =</td>
<td>7</td>
</tr>
<tr>
<td>&gt; =</td>
<td>8</td>
</tr>
<tr>
<td>=</td>
<td>9</td>
</tr>
<tr>
<td>&lt; &lt; =</td>
<td>10</td>
</tr>
<tr>
<td>NOT</td>
<td>11</td>
</tr>
<tr>
<td>AND</td>
<td>12</td>
</tr>
<tr>
<td>OR</td>
<td>13</td>
</tr>
</tbody>
</table>

This is more natural than that used in Pascal.

3.7 Miscellaneous
The fact that each statement starts on a separate line simplifies syntax checking and error recovery. This coupled with the fact that each statement is checked for syntax errors as it is read into the system and each segment (procedure, function or main body) is checked for static semantic errors when the end of the segment is reached, assists the user by enabling him to correct errors as the program is entered instead of permitting the user to make a number of errors and then attempting to unravel them all. Furthermore the dynamic debugging facilities available in BASIC viz. TRACE, BREAK, etc., have been included in BPL to make it easier for the user to debug his program.

4. Conclusion
The object in designing BPL was to produce a language which has, amongst others, the following properties:
(a) It is simple to learn and simple to use
(b) It is an interactive language with consequent benefits in the areas of syntax checking and debugging.
(c) It has neat control structures and lends itself to the production of readable, well-structured programs.
(d) It has a wide range of data types to facilitate programming a wide variety of applications.

At the end of the exercise it was discovered that a number of the facilities specifically excluded from Pascal because they might invite inefficient programming, had been included in BPL, whereas the dreaded GOTO (excluded from BPL) has been included in Pascal as a compromise to efficiency. Thus one might sum up by concluding that BPL places less stress on efficiency and more on assisting the programmer to produce a program which is readable and well-structured.

References
Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles or articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be in double-spaced typing on one side only of A4 paper and submitted to Dr. D. S. Henderson or Prof. M. H. Williams at Rhodes University Grahamstown 6140 South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:


Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of 'Letters to the Editor' (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.
Questiones Informaticae


Contents/Inhoud

Toepaslikheid van 'n analitiese model by die keuse van 'n leerstruktuur vir 'n teksverwerkingsstelsel ...... 1
A. Penzhordn

Direct FORTRAN/IDMS interface (without the use of a DML) on the ICL 1904/2970 computers, using a geological data base as example ................................................................. 16
B. Day

Rekenaarondersteunde onderhoud van programmatuurstelsels. ................................................................. 23
E. C. Anderssen

Database design: choice of a methodology ................................................................. 23
M. C. F. King, G. Naude, S. H. von Solms

Design principles of the language BPL ........................................................................ 26
M. H. Williams

A high-level programming language for interactive lisp-like languages .................. N/A
S.W. Postma

The sequence abstraction in the implementation of EMILY ........................................ 30
D. C. Currin, J. M. Bishop, Y. L. Varol

Block-structured interactive programming system .................................................. N/A
C. S. M. Mueller

The management of operating systems state data .................................................. 34
T. Turton

From slave to servant ......................................................................................... N/A
G. C. Scarrott

Teacher control in computer-assisted instruction ................................................ 34
P. Calingaert

The impact of microcomputers in computer science ............................................. 38
K. J. Danhof, C. L. Smith

Architecture of current and future products ......................................................... N/A
C. F. Wolfe

An algorithm for merging disk files in place ....................................................... 41
P. P. Roets

An algorithm for the approximation of surfaces and for the packing of volumes .......... 44
A. H. J. Christensen

The memory organisation of large processors ..................................................... N/A
D. M. Stein