Q. Is there any problem of incompatibility between scratch files used by different parts of a multi-language program?
A. Not on ICL System 4, provided that a file to be used from two languages is defined with the same record-form, blocksize, etc., in the appropriate shells. One physical file, with an actual filename, can be accessed in a program as one or more logical files, each with a different logical file-name – tie-up between physical and logical files is via JCL. Normally only one of these logical files would be open at a time – so, for example, an ALGOL module might write a disk file, known to it as A120 say, and close the file, which could then be read by a FORTRAN module as DSET45 (which would have to be defined appropriately in DATAD).

There arose a need to extend the BASIC language by adding stacking and character manipulation facilities. This was to meet the requirements of a group of users who were not professional programmers and whose knowledge of computing was confined to the BASIC language. To satisfy this need, a set of macros was written and used in a prepass to the BASIC compiler. This paper discusses the relative merits of using a macro processor in such situations.

Extending high-level languages by macros – a practical case evaluation

P.J. BROWN
Computing Laboratory, University of Kent at Canterbury

There have been many published descriptions of macro processors that can be used to extend high-level languages. These include general-purpose macro processors that can be applied to any high-level language and macro processors tailored to particular languages. However, there have been few detailed examinations of the applications of these macro processors. The purpose of this paper is to attempt to fill this gap by describing a practical usage of a macro processor to extend a high-level language. An attempt will be made to evaluate the advantages and disadvantages of this approach.

The application arose thus. A group of users wished to write programs that involved character manipulation and stacking. These users were not experienced programmers; their knowledge of programming was limited to the BASIC language. The BASIC compiler that was available [1] catered only for basic BASIC; it was not one of the recent tarted-up BASIC compilers that have made BASIC look more like ALGOL. It therefore contained no direct facilities for character manipulation nor any for stacking. The users did not want to spend a lot of time learning about some other programming language that would be more suitable for their problem. Hence it was decided to make the best of the BASIC compiler as it stood. Stacking operations can be programmed in terms of LET and IF statements, and character manipulation can be achieved by pretending that characters are numbers, each character being represented by its internal code. Doing this is, however, almost certain to lead to coding bugs, and programs become tedious and hard to understand. (If BASIC had an ALGOL-like subroutine facility, the situation would be better.)

This is, in fact, an example of a common problem. A language does not contain all the features the users want, but the users are naturally reluctant to learn a new language. The problem arises with all computer languages, though if the language is a simple one like BASIC, trouble occurs sooner than it would with, say, PL/1. The problem can often be surmounted by using a macro processor to extend the language in the way required by the particular users. In the situation considered here this is exactly what was done. By making a prepass through a macro processor, BASIC was extended to provide facilities to make it a suitable language for manipulating stacks and characters. The extended form of BASIC was called NEWBASIC.

The following is a list of the extra statements that are available in NEWBASIC.

1. Character input. The statement

INCH <u>variable</u>

inputs a single character and places it in the <u>variable</u>.

1a. Character input with user-controlled stop. The statement

```
INCH variable GONONE statement number
```

is the same as (1) above except that if data is exhausted control passes to the given statement number.

2. Clearing an input line. The statement

```
CLEARLINE
```

makes the next INCH statement take input from a fresh line.

3. Character printing. The statement

```
OUTCH expression
```

outputs the character whose internal code is the value of the expression.

4. Stacking. The statement

```
STACK expression
```

places the value of the expression on the top of the stack.

5. Unstacking. The statement

```
UNSTACK variable
```

unstacks the value on the top of the stack and places it in the variable.

5a. Controlled unstacking. The statement

```
UNSTACK variable GONONE statement number
```

is the same as (5) above except that, if the stack is empty, control passes to the given statement number.

6. Printing the stack. The two statements

```
PRINTSTACK
```

and

```
OUTCHSTACK
```

print the contents of the stack as a sequence of numbers and as a sequence of characters, respectively.

7. Clearing the stack. The statement

```
CLEARSTACK
```

clears the stack.

8. Setting the stack size. The statement

```
STACKSIZE integer
```

sets the size of the stack as the value of the integer. This statement is optional; if it is omitted a stack size of twenty words is assumed.

9. Testing for digits. The statement

```
DIGIT variable THEN statement number
```

tests if the value of the variable is the internal code for a digit. If it is, control goes to the statement number and the value of the variable is reset to be the numerical value of the digit.

10. Testing for letters. The statement

```
LETTER variable THEN statement number
```

is similar to the DIGIT statement, except that it tests for letters rather than digits.

In addition, NEWBASIC contains an extra facility for dealing with character constants. They are represented in the conventional way be enclosing the character within quotes, e.g.

```
STACK ")"
```

or

```
IF X = "+" THEN 200
```

(The internal codes for layout characters are represented by suitable names, e.g. NEWLINE.) Thus the user does not need to know the internal codes for characters.

Example

The following is an example of a complete program in NEWBASIC. As can be seen the new statements are written in an identical way to the existing ones and blend naturally with them.

The sample program calculates the values of arithmetic expressions containing digits, parentheses, addition operators and subtraction operators. For example if the data were

```
(9-5) - (6-7+(1-2))
```

the program would print

```
ANSWER IS 6
```

Each line of data contains a single expression. If there is an error in the data an appropriate message is printed. The program could clearly be extended to cater for more general expressions.

```
10   CLEARSTACK
20   CLEARLINE
30   STACK -1        end marker
40   LET V=0         V is current left-hand operand
50   LET A="+"       A is current operator
60   INCH X GONONE 999(To 999 if no data lines left)
70   DIGIT X THEN 200
80   IF X NE "(" THEN 920   Error in data
90   STACK V  Stack previous operand and operator
100  STACK A
110  GOTO 40

200  REM CASE OF OPERAND
210  IF A= "-" THEN 240
220  LET V=V+X
```

```
230  GOTO  300
240  LET  V=V-X

300  REM  EXPECTING  OPERATOR  OR  ")"
310  INCH  A
320  IF  A= "+"  THEN  60
330  IF  A= "-"  THEN  60
340  IF  A= NEWLINE  THEN  500  End of line
350  IF  A  NE  ")"  THEN  910   Error in data
360  LET  X=V
370  UNSTACK  A Unstack previous operator and operand
380  UNSTACK  V
390  GOTO  210

500  REM  END  OF  LINE
510  UNSTACK  T
520  IF  T  NE -1  THEN  900  Unmatched parentheses
530  PRINT  "ANSWER =";V
540  GOTO  10

900  REM  CASE  OF  ERROR
910  LET  X=A
920  PRINT  "DATA ERROR:  OFFENDING  CHARACTER=";
930  OUTCH  X
940  PRINT                  Output the line
950  GOTO  10

999  END
```

The mapping

The mapping of most of the NEWBASIC statements into BASIC is quite straightforward. The stack is simulated within a BASIC array, and NEWBASIC stacking statements map into BASIC statements to adjust the stack pointer, check for overflow or underflow, and then move the data item to or from the stack.

For character I/O the BASIC compiler has a library package which contains two built-in functions that provide single character input and output, characters being treated as numbers (their internal codes). As they stand these are not absolutely straightforward for the beginner to use as it is necessary to load the library package specially and to specify certain function parameters. However they provide the primitive operations into which the INCH and OUTCH statements of NEWBASIC can be mapped.

The mapping is accomplished by a prepass through the ML/I macro processor [2]. After this prepass the NEWBASIC job behaves exactly as an ordinary BASIC job.

Control cards

The control cards for running a NEWBASIC job are no more complicated than those for an ordinary BASIC job. The control card at the start of a NEWBASIC job invokes a predefined sequence of commands that performs the macro prepass and then enters BASIC.

This is an important point. It is wrong to claim that a language is usable by beginners if it is necessary to supply a lot of extra control cards to invoke the language.

Problems

So far, the discussion of NEWBASIC has been confined to its positive features. It is now time to consider some of its problems and disadvantages.

Firstly, it is necessary to place some restrictions on the use of BASIC. There are, in fact, no major restrictions but a number of small ones. Most of the restrictions arise because certain variables, statement numbers, functions, etc. are reserved for use in the mappings of the NEWBASIC statements. For example, all the variable names beginning with Z are reserved, Z1 being the stack pointer, Z2 a buffer pointer, etc. For similar reasons statement numbers greater than 20000 are forbidden. These restrictions are necessary because BASIC has no block structure and no facility for local scope.

Another restriction, and perhaps the most serious one, arises because of the requirement in BASIC that statements be numbered in increasing order. The statements in NEWBASIC map into up to three BASIC statements. For example,

```
150     UNSTACK  X
```

maps into

```
150     IF  Z1  LE  0  THEN  20060  (Error:
                                    stack is empty)
151     LET  X=S(Z1)
152     LET  Z1=Z1-1
```

As can be seen, these statements are numbered sequentially from the original statement number, and the user must not use the newly-introduced statement numbers in his own program or there will be a clash. Hence NEWBASIC users are advised to leave a gap of three between all their statement numbers. This restriction is due to the nature of BASIC and would not, of course, be necessary in many other programming languages.

Operational environment

Like BASIC, NEWBASIC can be run either on-line or in the batch. It is, however, less interactive than BASIC. In BASIC, errors are detected as a line is typed in. In NEWBASIC the entire program is typed in, then macro processed and then passed through the BASIC compiler.

There are two possible kinds of syntactic error in NEWBASIC programs:

1. Errors detected during the macro prepass.
2. Errors detected during the BASIC compilation.

The mapping macros contain very little checking, and therefore errors are usually of type (2). It is easy for the user to correct type (1) errors, given that the macro processor gives comprehensive error messages. Type (2) are more difficult. Assume, for example, that the user writes

```
150   UNSTACK  Z3
```

This maps into

```
150   ...
151   LET  Z3 = S(Z1)
152   ...
```

The BASIC compiler therefore gives a message that there is an illegal assignment in line 151. The user needs to relate this back to his original NEWBASIC program, which, of course, contains no line 151. Hence the user needs to look backwards from the given line number and diagnose the true cause of the error for himself. (Alternatively he can look in detail at the BASIC program that his NEWBASIC program has mapped into, but this should only be a last resort.)

Happily this problem was, in practice, not as bad as it appeared, and users were able to diagnose their errors without any help. Perhaps this was because they had sufficient programming experience to be able to do a little detective work to resolve errors.

Conclusion

On the basis of this experience it is felt that the advantages of the approach described outweigh the disadvantages. Macro-extended high-level languages are not only of value to the professional programmer but can make computer usage easier for the comparative beginner as well. The range of application is huge, as the requirement for languages oriented towards special purposes arises in most environments.

References

1. BINNS, S.E., 'Kent On-line System: the BASIC compiler', University of Kent at Canterbury (1970).
2. BROWN, P.J., 'The ML/I macro processor', Communs. Ass. comput. Mach., V10(10), (October 1967).

Discussion

Q. Concerning your restriction of variable names, why didn't you delve into the computer? Then the main processor could accept everything. Have you any estimate of the programming efforts involved?

A. I do not believe it is wise to try to change compilers unless there is no alternative. The programming effort in writing the ML/J macros for NEWBASIC was less than a man-day.

Q. What losses would there be if you had a subroutine to do the stack facility?

A. In BASIC the notation for calling subroutines with parameters is very clumsy. In ALGOL, for example, there would have been no need for a stack macro, as users could have written a subroutine call in an equally convenient notation.

Q. What language is the macroprocessor written in? Can you give further details?

A. It is written in a machine-independent low-level language called LOWL. It is reasonably easy to implement this on any computer. Someone here claims to have done it in four days for one computer.

Q. One advantage of macros in an assembly language is that you can separate the algorithm. Is this true in a high-level language?

A. I think there is no inherent difference between the use of a macroprocessor with a high-level language and its use with an assembly language.

Q. What about macros within macros (nesting)? Can you comment?

A. You can certainly do this in ML/I, and a form of nesting arose in this application, e.g.

```
STACK "X"
```

Q. I like the macro facility. How difficult would it be to implement in JCL?

A. ML/I can be used as a pre-processor to JCL if the operating system permits this.

Q. Can you substitute a statement number in a macro statement?

A. Yes. This was done in NEWBASIC by defining the state of a line as a macro and taking the statement number that followed as its argument. This macro 'remembered' the current statement number so that other macros could use it.

Q. Were these students using the system doing problem solving or merely computer-aided instruction?

A. Problem solving.

Q. Are there any error messages in ML/I?

A. Yes. In particular the NEWBASIC macros give the statement number of the offending statement.

Q. I notice your macroprocessor has conditional statements. What are your views of this? How valuable is it without?

A. If a macroprocessor lacks a conditional capability its usefulness is limited. If you want this roughly quantified, I would say its power is halved.

Q. What happens if you require a listing of the Macros?
A. This is easily obtained.
Q. Why didn't you check overflow?
A. I checked for overflow on STACK and underflow on UNSTACK.

Q. How do you handle and generate the print statement at label 22400 to obtain the error message?
A. There is a set of BASIC statements that the macros automatically tack on to the end of each NEWBASIC program.

Q. How fast is the ML/I in computing?
A. A rough figure is 3000 macro calls a minute on a machine with a 2-μsec store. When ML/I is performing a complicated translation (NEWBASIC is very simple) it may take 10-50 macro calls to process each line, and it is in these cases that it appears slow.

Q. Do your users in general use your macros or write their own?
A. A systems programmer usually writes the macros.
Q. Can you bring new macros into ML/I and then combine them?
A. Yes.

The input of data containing directives, comments, variable length character strings, or having a nested or otherwise complicated structure can not be dealt with conveniently by the standard input procedures of FORTRAN and ALGOL. Data definitions, called phrase structures, have been added to 1900 FORTRAN and ALGOL to simplify the writing of input procedures for such information. This paper describes the main features of the data definitions and illustrates their use with reference to the specification of data formats for an ICL applications program package. The phrase structure implementation is by means of a preprocessor which translates phrase elements into FORTRAN or ALGOL instructions.

Input of structured data in FORTRAN and ALGOL

DR R.J. HOUSDEN
University of East Anglia, Norwich

This paper is concerned with special facilities designed to simplify the input of data prepared in variable formats. Such data might include:

1. Comments which are to be ignored by the program.
2. Directives which invoke appropriate procedures to deal with subsequent data items.
3. Items of variable length such as text.
4. Items having a nested or otherwise complicated structure.
5. Simple numeric data punched in free format.

An example of input routines which deal with data having a non-trivial structure is provided by an ICL Power System Analysis package. The programs were written in FORTRAN with a few PLAN subroutines. One suite of programs is concerned with a.c. load flow studies [1]. The network data consist of a number of data directives each followed by a block of data records. Each directive and its associated data block may appear any number of times anywhere in the data. If an item of data is specified more than once, then the last value will overwrite previous values. Any number of blank cards, or newlines, may be included in the data to improve the layout of the optional listing on the line printer. Each directive starts in column 1, or at the beginning of a line. Only the first four characters of any directive are significant, the remainder of the record being ignored. Numeric values in data blocks are punched in free format. Each value may be preceded by and is terminated by one or more spaces or blank columns, except where a numeric value is followed by a name, in which case the value and the name are separated by a single space.

This example is typical of many situations in which the standard FORTRAN input package is inadequate for the following reasons.

1. Data directives are followed by an indefinite number of data records.