On "EASE", an Automatic Programming Prepared for NEAC-2203 Digital Computer

YASUO TAMURA*, TORU KIDERA** AND KEN MASEGI***

1. *Introduction*

"EASE" stands for *Efficient Autocoder for Saving Efforts.* The grammar of "EASE" is very similar to that of FORTRAN.

NEAC-2203 is a transistorized decimal computer with a 2000 word drum memory with quick access bands, each word consisting of eleven digits plus a sign bits. 4000 instructions or 2000 data can be stored in the drum. Average execution time for arithmetic operations is about 6.7 ms including the access time.

2. *Sample Program written in "EASE"*

Let us show a source program written in "EASE" before describing the grammar. Sample program: Computation of square root with Newton's method of successive approximation.

Source program:

```
    VARIABLE
    FLOAT, X, 2, A, 1,
    FIX,                              ¥
 4  FORMAT (9, 2, 2)                  ¥
 3  READ (1) A,                       ¥
    X ♯. 1E 20                        ¥
 1  X 1 ♯ (A/X+X)※.5                  ¥
    IF (ABSF(X-X 1)-. 1E-10) N 2      ¥
    X ♯ X 1                           ¥
    JUMP 1                            ¥
 2  WRITE (1, F 4) A, X,              ¥
    STOP 3                            ¥¥
```

Table, prepared by "EASE" processor:

```
.  S
0  1      0 0 0 3 X
0  2      0 0 1 2 X
0  3      0 0 0 0 X
.  D      0 0 1 6 X
.  T      0 0 2 0 X
A         0 0 2 2 X
X         0 0 2 3 X
```

Object program, generated and punched by "EASE" processor:

```
ZA
7730100 Y   1000000 Y   1100022 X   1900017 X   1100023 X
1000000 Y   5700000 Y   3000022 X   1700023 X   3008802 Y
2000023 X   1408800 X   3200018 X   1100024 X   3000023 X
```

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 3, No. 3 (1962), pp. 127–134.

* Electrical Engineering Department, Waseda University, Tokyo. ** Fuji Denki Seizo Co., Ltd., Tokyo. *** Tokyo Shibaura Denki Co., Ltd., Tokyo.

```
2100024 X    1103340 X    3403340 X    2100019 X    4100012 X
1900024 X    1100023 X    4300003 X    1000000 Y    1900016 X
1100219 Y    1300022 Y    7730200 X    1300023 X    7730200 Y
4400000 X    1000000 Y    0016 Z G
000000090202 Z I      070100000000 Z I      050500000000 Z I
040100000000 Z I      4400000 Z E
```

3. *General Description of "EASE" Statements*

3.1. *Statement numbers (abbreviated as S. No.)*

Numerals 1 to 70 can be assigned to S. No. (if violated, error 1).

3.2. *Constants*

An integer (fixed point) should be 11 decimal digits long or shorter without a decimal point. The mantissa of a fractional number (floating point) is 9 decimal digits long or shorter with a decimal point, while the exponent should be between −51 and 50. When more than one constant with the same value are used in a source program, only one of them will be recorded as a representative.

Examples of integers: 0, 15, 105, 99 999 999 999.

Examples of fractional numbers: 20.0, 20., 0.2, .0, 12.34 E3 (=12340.0), 12.34 E-3 (=0.01234).

3.3. *Variables*

(1) Simple Variables:

a) Declared Variables: Only first five characters are kept in memory and a tail portion will be discarded.

b) Undeclared Variables: IX1, IX2 and IX3 indicate index register No. 1, 2 and 3, respectively. Index register No. 3 takes part in subroutine jumps. Index register No. 2 is used in a LINK-END statement.

(2) Subscripted Variables: Subscripts may be added to a declared variable. A subscript should be an integer in nature (see ex. 1 below). A subscript is bracketed with ( ) as a rule (ex. 2). An array expression is not defined in EASE. There is no restriction on the length and the number of nesting of a subscript (ex. 3).

ex. 1. A(TRUNF(B+C)), A(ROUNDF(B)−I), where B and C are floating point variables and I, a fixed point variable.

ex. 2. A(0), AO and A are regarded as the same.

ex. 3. A(I(J(K+3))), A((I−1)※N+(K−1)), where I, J, K, and N are fixed point variables.

3.4. *Functions*

(1) Function Evaluation by Subroutines: The following functions are

evaluated with the aid of subroutine tapes, especially prepared for the EASE compiler: SQRTF(X), SINF(X), COSF(X), ARTANF(X), SINHF(X), COSHF(X), LNF(X) ($=\log_e(X)$), LOGF(X) ($=\log_{10}(X)$), EXPF(X) ($=e^X$), POWER(X, Y) ($=X^Y$), where X and Y are floating point variables. A function can be an argument of another function. There is no restriction on the length of an argument.

ex. SQRTF (SINF(A+B)+COSF(LNF(C))).

(2) Function Evaluation by Commands: The following functions are evaluated with the use of appropriate commands in the target program: ABSF(X), FLOATF(X), TRUNF(X), ROUNDF(X).

3.5. *Symbols (except alphabets and numerals)*

Four arithmetic operators (+, −, ※, /) are permitted. A minus sign (−) is also used in a negative exponent of a constant. Comma (,) separates variables or constants. "¥" is used as the statement end mark and the program end mark. Therefore, (···¥¥) appears at the end of a source program.

3.6. *Length of a statement*

A statement is 90 characters long or shorter with "¥" included. But a variable statement may be as long as needed. Since the memory space for an old working storage is automatically assigned for a new working storage, as soon as the former gets unnecessary, only one working space will be require for such a combination of parentheses as (--(--(---)--)--).

ex. $\underset{7}{(}\ \underset{6}{(}\ \underset{5}{(}\ \ \underset{5'}{)}\ \underset{6'}{)}---\underset{4}{(}\ \ \underset{4'}{)}\ \underset{7'}{)}---\underset{3}{(}\ \underset{2}{(}\ \ \underset{2'}{)}\ \underset{3'}{)}---\underset{1}{(}\ \ \underset{1'}{)}$. The processing goes on in ascending order of numbers. Four working storages are required for the example. 14 working storages are sufficient for most expressions.

4. *Detailed Description of "EASE" Statements*

4.1. *Arithmetic statement*

This takes the form of [Variable]♯[Arithmetic Expression]¥. Arithmetic operations (+, −, ×, /) between floating and fixed point numbers are not permitted, and will cause an "Error 7" halt. Either one of floating or fixed point expression as well as a variable of the same type should be used on both sides of a statement.

4.2. *Logical statements*

(1) JUMP Statement: "JUMP 5¥" means an unconditional jump to S. No. 5.

(2) STOP Statement: "STOP 5¥" stops the computation. Restarting will transfer the control unconditionally to S. No. 5.

(3) IF Statement: Expressed as "IF([Arithmetic Expression]) P[S. No.] Z[S. No.] N[S. No.]¥." P, Z or N may come in a random

order. Not all of P, Z and N need be written.

ex. IF (A−B) P5 N6 ¥ or IF (A−B) P5 N6 ¥ stands for a conditional jump to S. No. 5, if A−B>0, or to S. No. 6, if A−B<0, respectively.

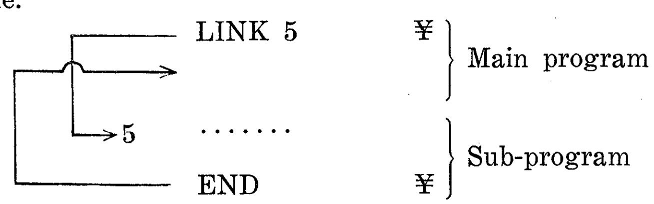
(4) ITER Statement (iteration): ITER ([S. No]) [Variable], [Variable or Const. (Increment)], [Variable or Const. (upper limit)], ¥.

The initial value is set before a loop begins, and ITER statement will follow the loop statements.

```
              1#[Initial value] ¥
5  .......                   ¥ }
                             : } loop statements
    ITER(5) I, J, K          ¥ }
```

(5) LINK Statement: A portion of a source program will be used as a subroutine with the aid of a LINK statement, combined with an END statement.

(6) END Statement: Index register No. 2 is used with a LINK statement. "END ¥" may be used with the Runge-Kutta-Gill subroutine.



(7) Subroutine Statement: This statement connects a subroutine having some parameters with a main program. It is written in the form of SR [address] ([parameter], [parameter], ---, ---) ¥.

An address next right to SR specifies the first location of a subroutine, to which a "Set Index (3) Jump" will be executed from the main program.

ex. 1. SR1000(5, A, B) is a general form of matrix subroutines.

ex. 2. SR500(N, .S5, H, Y, X, DY, Q) is a form for Runge-Kutta-Gill subroutine.

4.3. *Declaration statements*

(1) Variable Statement: This statement should come first in a source program.

ex. VARIABLE
FLOAT, TEMPERATURE, 3, PRESSURE, 4,
FIX, COUNT, 5, ¥

(2) FORMAT Statement: This takes the form of FORMAT ([integer], [integer], [integer]) ¥. A FORMAT statement should be headed with a S. No. which will be referred to by any WRITE statements

coming later in order to select the print format. The first, second and third integers from left to right specifies (i) the length of data words (mantissa in case of floating point), (ii) the number of spaces which will be inserted between adjacent data words, and (iii) the number of data words per line, respectively.

ex. 6 FORMAT(9, 2, 3) ¥.

(3) CODE statement: This permits us to write a source program in an assembly language. Headed with a single CODE statement, up to 50 mnemonic codes could be placed without "¥" symbol after each code.

ex.
```
CODE ¥          ┐
CHR 0 1003,     │
HRS 0 2,        ├ group of mnemonic codes
UCJ 2 .S5,      │
NET 0,  ↑   ¥   ┘
 ↑   ↑  │
 │   │  └─ address
 │   └─ index register
 └─ operation code
```

4.4. *Input-output statements*

(1) READ Statement: Written as READ(1) X(I), Y(J+I), K, K(I−L).

(2) WRITE Statement: An output format is specified by the statement number after "F" in the WRITE statement. Refer to FORMAT statement.

ex. WRITE(3, F3) X, I, ¥.

(3) AWRITE Statement: With this statement, a string of characters will be typed out, with "¥" excluded.

ex. "AWRITE(1) EASE ¥" will type out "EASE".

4.5. *Mode statements*

"EASE" processor starts compiling in "Floating off" mode, and remains in the same mode except when needed as in case of Add, Subtract, Multiply and Divide.

(1) FIX Statement: "FIX ¥" will generate a "FOF" command.

(2) FLOAT Statement: "FLOAT ¥" will generate a "FON" command.

4.6. *Break point statement*

"BKP ¥" will generate a Break Point command.

5. *Error Indication*

When a grammatical error is found, the type of error and the whole statement which contains it will be typed out for inspection by a programmer. There are 11 types of errors indicated.

6. *Conclusion*

Since one of the authors used an IT compiler (Internal Translator) for IBM-650 at the University of Michigan in 1957, and also a FORTRAN for IBM-704 at General Motors Company, Detroit, Michigan in 1958, it has been our desire to make an automatic programming system by ourselves.

A Block Diagram Simulator [1], "SWEEP", was completed for an LGP-30 in January, 1960, which is a simulation of an analog computer by a digital one. This program has had a wide field of applications since then, especially in the field of automatic control.

This was succeeded by "EASE". EASE compiler was completed in March, 1962. It took us about six months.

We should like to acknowledge the continued encouragement and support of Professor Masato Namba, Director, and the members of Computation Center, Waseda University.

Reference

[1] TAMURA, Y., Combination of a Block Diagram Simulator "SWEEP" and Automatic Programming System (in Japanese). *Journal of the Institute of Electrical Engineers of Japan*, July, 1960, 923 930.