DARTMOUTH

OVERSIMPLIFIED

PROGRAMMING

EXPERIMENT

Dartmouth Computation Center
May 1962

Preface

The Dartmouth Oversimplified Programming Experiment (to be referred to by its initials) was designed for a dual purpose. It will first of all be used in an experiment designed to teach a large number of freshmen the rudiments of programming in a course of three one-hour lectures. For this experiment DOPE provides a fast and efficient method of running programs written by the students. It is also hoped that DOPE will prove sufficiently convenient to serve for programming of simple research problems.

The language of DOPE was designed by Professor John G. Kemeny, with the following principles in mind: (1) It is a universal language. That is, any problem that the machine is capable of handling can be written in DOPE. (2) Each instruction in DOPE corresponds to a box in a flow-diagram. Hence it is trivial to translate a flow-diagram into a DOPE program. (3) The language is so designed that it can be quickly translated into machine language, and hence the usual lengthy compiling time is avoided when the program is actually run. To achieve these goals, some flexibility had to be sacrificed. However, the rigidity of the language actually makes it easier to teach to the novice.

A programming language is useful only if it can be translated into machine language. The compiler (translator) for DOPE compiles any program for the LGP-30 almost as fast as the flexowriter can read it. One minute is a typical compiling time. If the DOPE program and its data have been pre-punched on a tape, the machine will immediately after compiling start its computation, and will run the problem for as many sets of data as have been provided. The running time of a program is roughly found by allowing one minute for 200 arithmetical operations.

The DOPE compiler was written and tested by Sidney Marshall '65, a freshman research assistant at Dartmouth.

Flow diagrams.

A flow-diagram is a simple device for stating precisely a set of computational rules. Four types of components occur in a flow-diagram: Arithmetical, comparison, loop, and input-output. In addition, there are arrows to show the order of instructions.

Arithmetical boxes.

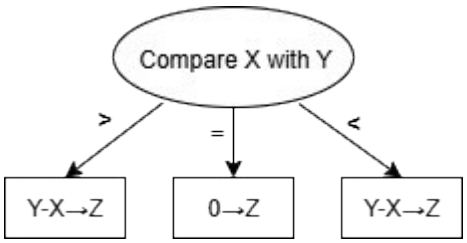
Each such box contains the order for one arithmetical operation. For example:

$X + Y \rightarrow Z$	Compute X + Y and call the result Z
$2/Y \rightarrow W_3$	Divide Z by Y and call the result W_3
$SIN(X_5) \rightarrow X_5$	Compute the sin of X_5 and call the result X_5
$X \rightarrow Y$	Let Y equal X.

An arithmetical order is indicated by enclosing it in a rectangular box.

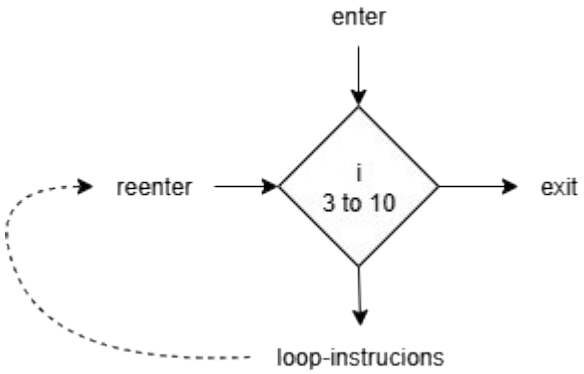
Comparison.

A comparison is of the form: Compare X_2 with Y. The comparison results in one of three decisions: $X_2 > Y$, $X_2 = Y$, or $X_2 < Y$. The comparison is placed inside a circle, and it has three arrows coming out of the circle, corresponding to the three possible results. For example, in the following diagram we compute $Z = |X - Y|$:



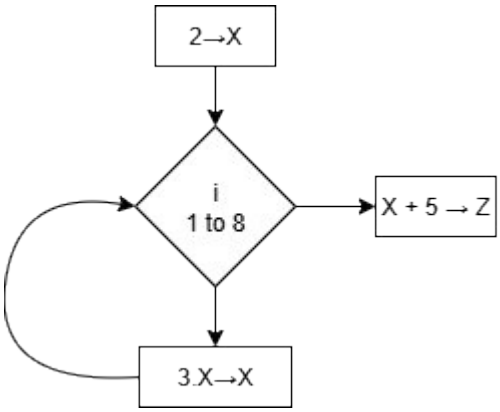
Loop.

A loop is a very powerful device. It serves to carry out a sequence of orders repeatedly, for varying values of an index or a parameter. The key order of a loop is enclosed in a diamond:

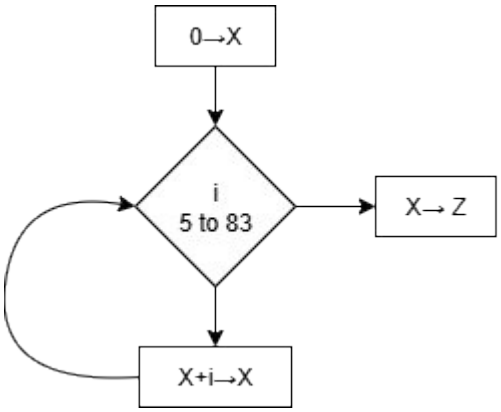


One enters the diamond on top. It then sets I = 3, and proceeds to whatever instructions are to be carried out in the loop. When the diamond is reentered (from the left) the value of I is increased by 1, and the loop-instructions are repeated. When the instructions have been carried out for I=10, then the loop is 'exited' to the instruction on its right.

For example, the diagram below computes $Z = 2 \cdot 3^8 + 5$:



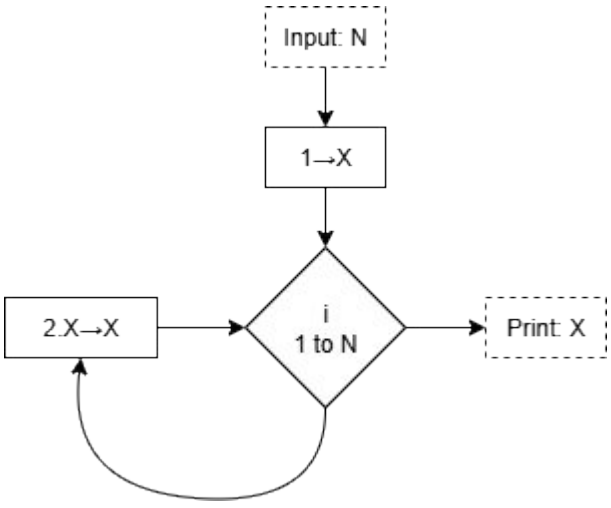
As another example, let Z = 5 + 6 + 7 + … + 82 + 83:



Input and output.

Most programs have small variable data. That is, some quantities may change from one run of the problem to another. For example, if one writes a program to compute a power of 2, one might as well compute 2^N, where N is to be specified as data. For this we need an input order. We also need a print order to have the flexowriter type an answer. These orders are indicated in dotted boxes, and usually occur at the beginning and the end of the program, respectively.

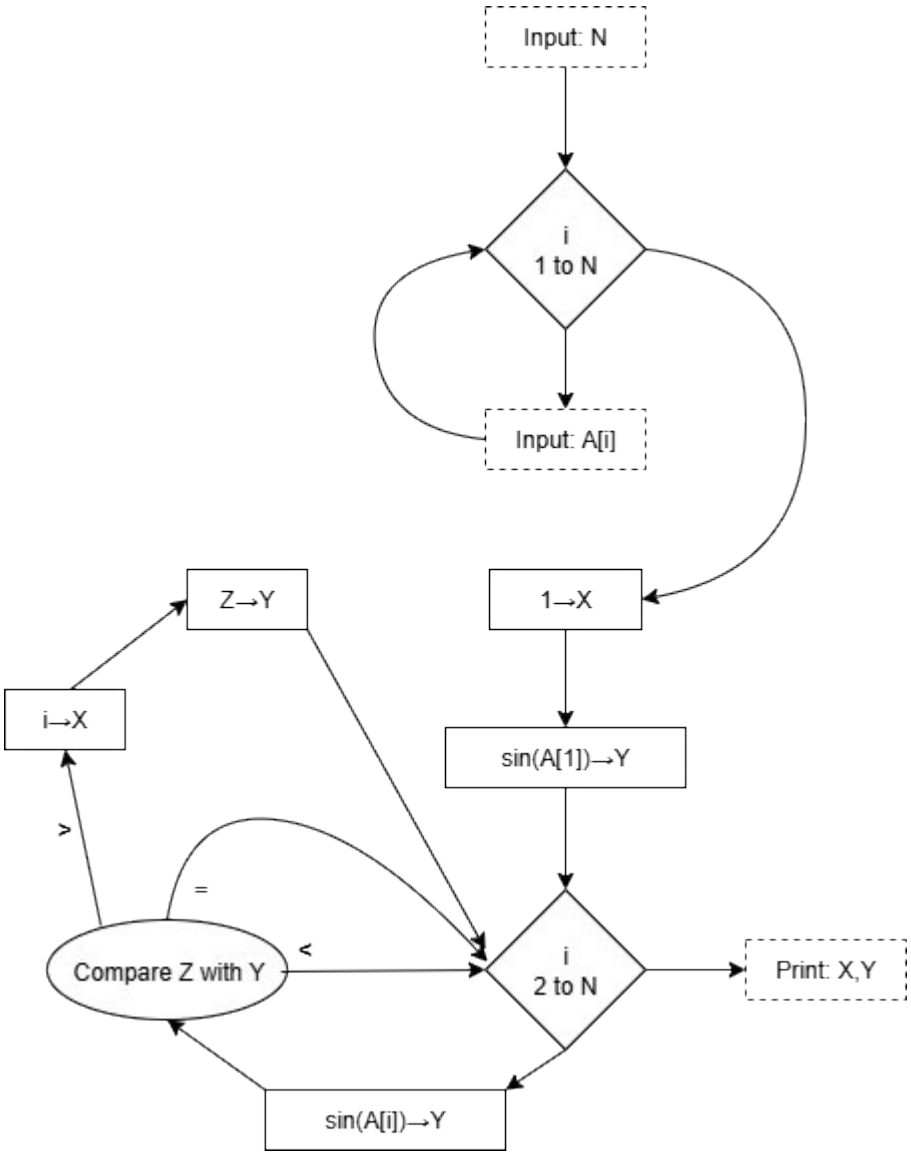
Consider the following example:



This program will first get a value of N from the data-tape, then it computes 2^N and prints the answer. Finally, it asks for a new value of N and repeats the procedure, until the data-tape is exhausted.

Illustration.

Let us construct one larger example. Suppose that we have a vector (or simply a set of numbers) A_I for I = 1, 2, ..., N. We wish to find the component A_I for which $\sin(A_I)$ is a maximum. In the following program both the index of such a component and the maximum value are printed:



DOPE INSTRUCTIONS AND FLOW DIAGRAM EQUIVALENTS

DOPE	FIELDS					Flow Diagram
Instr.	1	2	3	4	5	
+	A	B	C			A + B → C
-	A	B	C			A - B → C
·	A	B	C			A · B → C
/	A	B	C			A / B → C
;	A	B				A –→ B
SQR	A	B				√A → B
EXP	A	B				e^A → B
LOG	A	B				log(A) → B NATURAL LOG
SIN	A	B				sin(A) → B
C	A	B	L_1	L_2	L_3	Compare A with B; > = < ; L_1 L_2 L_3
T	L					——→ L
A						Print label
P	A					Print: A
N						Printer: Start new line
I	A					Input: A
Z	A	B	C			A B to C
E						End loop. (Reenter last loop, or go to next instruction if A=C.)
F						Stop! (To be used as final step for single data set)
S						Start computing. Must be on bottom of all programs.

Notes on DOPE instructions.

In place of A, B, and C in any instruction one may use any variable. For the purposes of DOPE a variable is a letter or a letter followed by a single digit. For example: A, X, Z5, Q0, M, T7 are variables. There is an exception: Do not use the letters L or O since these are indistinguishable from the numbers 1 and 0 on a typewriter.

One may also replace A, B, and C by a constant. A constant has 3, 4, or 5 characters. Each character is either a digit, or a decimal point, or a minus sign. For example: 1,0, 3456, -12.3, 001, -.001 are constants. Since a constant must have at least 3 characters, one writes 13.0 instead of 13. If the programmer wishes to use a more complicated constant, he should write a variable into his program, and then input its value as data.

In place of L, L1, L2, L3 in an instruction one must give the number of an instruction. This is the reason why all instructions are numbered. Instructions are numbered from 1 to 99, and hence one places a one or two-digit number in place of L. Thus, for example, the instruction T 17 directs the machine to take instruction number 17 next. The instruction C X2 Y 11 5 11 has the effect that if XZ < Y or XZ > Y then instruction 11 is taken next, if XZ = Y then instruction 5 is next. These instructions take the place of some of the arrows in a flow-diagram. There is no need to insert all arrows into the DOPE program, since the machine will take the next instruction in order, unless told otherwise.

There are 4 16-component vectors available in DOPE, denoted by E, F, G, and H, followed by brackets. Thus E[5] denotes the fifth component of E, more usually written as E_5. (The flexo-writer cannot type subscripts, hence brackets are used.) One may also write F[I] to indicate the Ith component of F, as long as the program determines the value of I to be between 1 and 16. Expressions like E[5] or F[I] may also be used in place of variables.

Inputting of data is necessary either when the value of a variable changes from one run of the program to another, or when one wishes to use a constant that is more complex than is permitted in DOPE for constants inside the body of the program. The data for a run of the problem is read in immediately after the program, and each piece of data must be called for by a J instruction. Each number inputted into the program as data has a magnitude and an exponent. The magnitude has a sign followed either by up to six digits, or up to five digits plus a decimal point. The exponent has a sign followed by 2 digits. The exponent indicates a power of 10, and must lie between -36 and +36. For example:

+12.36	+02	is	$+12.36 \times 10^{2}$	or	1236
-1.29	+20	is	-1.29×10^{20}		
+123456	-10	is	$+123456 \times 10^{-10}$	or	.0000123456
+297	+00	is	297		

Answers are printed by the flexowriter in the same format. Four numbers are printed per line. But the programmer may start a new line at any time by inserting an N instruction.

The programmer may, if he wishes, insert labels in the output. He simply puts an A instruction in a suitable place in the program, and writes a (short) label in the corresponding spot on the data sheet. When the program reaches the A, the label will be printed. If both J and A instructions occur in the program, care must be taken that the data and labels occur in the correct order. For example, if the instructions

```
J N
A
J M
```

occur in that order, then the data sheet should contain the data for N, followed by the label, and then the data for M. If there is more than one run, these three items must occur on the data sheet for each run. For example, if N = 10 and M = .023 for the first run, while N = 20 and M = -.345 for the second, the data-sheet may look as follows:

Run no.	Variable	Magnitude (or label)	Exponent
1	N	+10	+00
	Label	First answer	
	M	+23	-03
2	N	+20	+00
	Label	Second answer	
	M	-.345	+00
		FINISH	

Note: The flexowriter does not distinguish between capital and small letters. It is best to write programs entirely in capitals, for clear legibility. However, the typed program will be in small letters.

EXAMPLE 1. Square-root of integers up to 100.

Instr. No.	Instr.	F I E L D S		
1	A			
2	Z	A	1.0	100
3	SQR	A	B	
4	P	A		
5	P	B		
6	N			
7	B			
8	O			
9	S			

DATA SHEET

Run No.	Variable	Magnitude or label	Exp.
1	Label	Square root table	

EXAMPLE 2. **Illustration on p.5.**

Instr. No.	Instr.	Field 1	Field 2	Field 3	Field 4	Field 5
1	J	N				
2	Z	I	1.0	N		
3	J	A[I]				
4	E					
5	;	1.0	X			
6	SIN	A[1]	Y			
7	Z	I	2.0	N		
8	SIN	A[I]	Z			
9	C	Z	Y	10	12	12
10	;	I	X			
11	;	Z	Y			
12	E					
13	P	X				
14	P	Y				
15	F					
Last	S					

Data sheet starts with N, then lists A_1, A_2, ..., A_n in order.

EXAMPLE 3. $\int_0^1 e^{-x^2}\,dx$; N intervals, (trapezoid rule).

PROGRAM

```
 1    A
 2    J      N
 3    /      1.0    N      D
 4    ;      0.0    X
 5    ;      0.5    S
 6    Z      I      1.0    N
 7    +      X      D      X
 8    ·      X      X      Y
 9    ·     -1.0    Y      Y
10   EXP     Y      Y
11    +      S      Y      S
12    E
13    /      Y      2.0    Z
14    -      S       Z     S
15    ·      S      D      S
16    P      S
17    T      1
LAST
```

DATA

```
Run 1   Label   Problem 1
          N       +10          +00
        Label   Problem 2
          N       +100         +00
```

FINISH

Corrections:

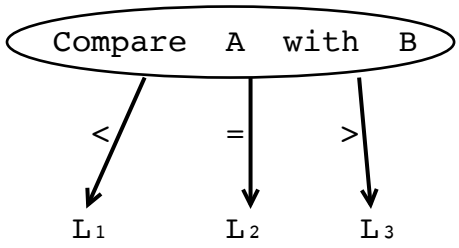
1./ A subscripted variable must have a variable as its subscript. That is, E[A] is legitimate and E[A5] is legitimate, but E[5] is not.

2./ A subscripted variable must not occur inside a Z instruction. That is

```
Z I E[A] 100
```

is not allowed.

3./ The C instruction is incorrectly listed on page 6. The correct order of the branch labels is:



Typing instructions for DOPE:

1. **Dope Sheet.**

Do not type instruction number. Type only instruction followed by 0 to 5 fields, each followed by '. For example, Example 1 on page 9 would appear as:

```
a'
z'a'1.0'100'
sqr'a'b'
p'a'
p'b'
n'
e'
f'
s'
```

Do not leave any spaces. Type one instruction per line. Remember to type s' at the end of the program. The data should follow the program immediately on the same tape.

2. Dope data sheet.

Do not type number of run or name of variable. For a label, type label as written (spaces allowed, but no stop code within label). Put one stop code at the end of the label. For data you must type two numbers of up to 5 characters each, followed by stop-codes. If the magnitude has no more than 5 characters, then make it the first item followed by a stop-code, and then type the exponent followed by a stop code:

5.297 +10	is typed as	5.297'+10'
.5 -08	is typed as	.5'-08'

Note that the exponent must have three characters consisting of sign and two digits. There is one convenient exception, +00 may be omitted. Thus

5.23 +00	is typed as	5.23'+00'
	or as	5.23''

If the magnitude consists of more than 5 characters (a maximum of 7 is allowed), then put a stop-code after the fifth, and type the remaining one or two characters, immediately followed by the exponent and a second stop-code. E.g.,

5.23456 +11	is typed as	5.234'56+11'
-111111 -11	is typed as	-1111'11-11'
123456 +25	is typed as	12345'6+25'

Note: To avoid round-off errors, while .00005 may be inputted as is or as 5-05, the latter will be much more accurate. Therefore, type: 5'-05'. The data-sheet must end with finish''. However, this must be entered as a number, not a title. So if the first item on the sheet is a label, type: 'finish''. If there happen to be n labels in a row, type n stop-codes, followed by finish'.

Instructions for using DOPE compiler.

The compiler is available on a single large paper tape labeled "DOPE COMPILER". This is a HEX tape, to be read in on the photo-reader, using 10.4.

Put 6-bit input down.

Load DOPE program with data into the typewriter tape-reader.

Depress "one operation", "clear counter", "normal", "start", just as if calling for 10.4. These steps will at any time cause the machine to compile a DOPE program. [The usual instruction in 0000 has been changed to u0300, to allow for immediate transfer to the DOPE compiler.]

The program will compile. If Brake-point 32 is DOWN, then the program is immediately executed, using the data on the data-tape. If the brake-point is UP, it stops after compiling. Push start when ready to run. (For example, the latter procedure allows manual inputting of data.)

To restart a compiled program, push the four buttons, then type s and push start. To compile a new program, just put it in the typewriter tape-reader and push the 4 buttons.

Error stops occur on: Floating point exponent overflow, printing of same (prints +. and stops), negative squareroot, more characters than allowed on data (types e), illegitimate command, negative logarithm (prints log), or if you forgot to put down 6-bit input!

At the end of using DOPE, put up 6-bit input. To restore 10.4: (1) Switch to manual input, (2) Type e0000, (3) Push "Fill instruction", (4) Type k01fj, (5) Push "One operation", and "execute instruction". Test it by pushing the 4 buttons as usual.