ALGAE I*

A Compiler for the IBM 704†

by

Edward A. Voorhees
Glenn L. Carter
Jeanne Hudgins
Chester S. Kazek
Karl G. Balke

University of California
Los Alamos Scientific Laboratory
Los Alamos, New Mexico

May 26, 1958
Revised - July 21, 1958

*This work performed under the auspices of the U. S. Atomic Energy Commission.
†This version is for use with a machine having: 8,192 words of core storage, 4 logical drums, 7 tape units, CAD instruction. A peripheral tape-to-printer or simulator is required.
1. Introduction

Discussions involving the subject of defining problems for interpretation and coding by known automatic-coding systems generally suggest that the techniques for stating the control (or logic) of the problem are frequently difficult to understand and difficult to use. It seems that the difficulty is one of discovering a suitable language with which to define problem control. In programming a problem for hand coding, the familiar flow diagram has been successfully used (when needed) to define the control of the problem. Such flow diagrams (as we draw them) cannot be presented directly to present-day computers. It is the purpose of this paper to propose a flow diagram representation using simple algebraic language which can be directly entered into the computer and to describe a compiler which accepts problems coded in this form.

The first part of this description concerns itself with the development of an idea which was earlier presented in a paper entitled "Algebraic Formulation of Flow Diagrams". The second section describes a compiler for the IBM 704 (called ALGAE I) which incorporates most of the techniques described in the first part. ALGAE I makes use of FORTRAN I to form the actual 704 code.

ALGAE I was written as a means to permit programmers to use and evaluate this idea of program control. As a result, any comments, criticisms, etc., from users of the system are not only welcome, but eagerly solicited.
2. General Description of the Algae Language.

2.1 Basic Concepts.

Assume we have a flow diagram such that it could be used to hand-code a problem for any stored-program computer. Suppose we remove from the boxes all equations, statements of input-output tasks and other statements not directly related to the control and logic of the problem, and list them, with identification, elsewhere as reference material. We do not include in this list statements and questions pertaining to loops, numerical conditions, or switch and trigger conditions. The items remaining in the flow-diagram would form a statement or "part-picture" of the control for the problem. We will attempt to translate this control statement into a statement which can be easily written and entered into a computing machine.

In order to illustrate the above discussion and extend the idea further, let us consider an elementary example. Suppose in a problem we wished to compute

\[ U = \sum_{i=1}^{10} x_i + \begin{cases} a & \text{if } \Sigma < 0 \\ b & \text{if } \Sigma \geq 0 \end{cases} \]

The corresponding flow diagram might be drawn as:

Removing the items not directly related to control would leave:

If a distinction is now made between loop ranges, such as the \( i \)-loop in the above example, and conditions for execution of equations, such as the \( U \) test above, we introduce the symbols

\[ I_1 : i = 1(1)10 \quad \text{(meaning I takes successively the values 1 through 10 in increments of 1)} \]

and \( C_1 : U \geq 0 \).

Our example would then be completely defined as follows:

-1-
Example 1

<table>
<thead>
<tr>
<th>Control</th>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\cdots E_1 + I_1 E_2 + C_1 (E_3, E_4) + \cdots)</td>
<td>(E_1 : U = 0)</td>
</tr>
<tr>
<td>(I_1 : i = 1(1)10)</td>
<td>(E_2 : U = U + x_1)</td>
</tr>
<tr>
<td>(C_1 : U \geq 0)</td>
<td>(E_3 : U = U + b)</td>
</tr>
<tr>
<td></td>
<td>(E_4 : U = U + a)</td>
</tr>
</tbody>
</table>

where \(C_1\) here is understood to mean "execute \(E_3\) if \(C_1\) is satisfied and execute \(E_4\) if \(C_1\) is not satisfied." If we were to define \(C_1^*\) as the negation of the condition \(C_1\), the last term in the control statement could be written as

\[\cdots + C_1 E_3 + C_1^* E_4 + \cdots\]

provided it is known that when \(E_3\) is executed \(E_4\) would not also be executed (as would be the case if \(\sum x_i = 1\) and \(b = -2\)). Clearly the \(C^*\) convention is not essential since the condition \(C_2 : U < 0\) could be used instead. The meaning of \(I_1 E_2\) is evident and will be discussed more generally below.

2.2 Definition of Symbols.

Let us now define more completely the set of control statement symbols. These definitions make no restrictions on the characters with regard to their use in equation writing since control statements are assumed to be handled separately from equations.

\(C\) (assumed to have a subscript) represents a single condition for a two-way branch. Several conventions, defined in section 2.5, have been developed which use logical combinations of \(C\)'s. \(C\)'s are used for stating all conditions, except those inherent in range statements such as end-of-loop tests. \(C\)-type conditions include tests of numerical, logical, console switch, and trigger conditions. A more generalized, n-way branch is under development. Once a \(C\) has been defined, it may be used repeatedly throughout the problem.

\(E\) (assumed to have a subscript) represents a single equation, a continuous and closed set of equations (whenever one is done, all are done), an input-output task, program stop, or any other
statement or closed set of statements not directly related to problem control. An \( E_i \) may also be used more than once.

\( G \) and \( H \) \(^\dagger\) (subscripted). This convention is designed to facilitate transfer of control. The symbol \( G \) (with a subscript) denotes a transfer to \( H \) (with the same subscript). There may be a many-to-one correspondence between \( G \) and \( H \).

\( I, J, K, L, M, N \) (subscripted) represent single range statements or loop definitions. \( I_i \) in Example 1 represents the range statement \( i = 1(1)10 \). More generally, the expression \( I_j \colon a = b(c)d \) means that \( I_j \) defines a loop in which the subscript \( a \) varies from \( b \) to \( d \) in steps of \( c \). These may be used repeatedly.

\( S \) (subscripted) represents a control statement. It allows the program control to be defined by many sub-control statements which are in turn connected by a single master control statement. An \( S_i \), once defined, may be used only one time. This partitioning of the code into smaller units of control tends to clarify the various phases of the problem.

\( T \) (subscripted) is basically an \( S \) which may be used repeatedly within a problem. It is the Algebraic representation of a closed subroutine. Both the \( S_i \) and \( T_i \) can appear in the same positions as an \( E_i \).

The plus sign, \( (+) \), has two meanings. The first is to indicate a direct flow of control, as, for example, \( E_1 + E_2 \), which means "execute \( E_1 \), then \( E_2 \)". \( E_1 \) and \( E_2 \) are terms of the control statement \( E_1 + E_2 \). A second meaning, when used with C's, is discussed in section 2.5.

The parentheses, \( (,) \), are used for the phrasing or grouping of terms, for indicating ranges, and for special purposes to be described. The expression \( (E_1 + E_2) \) is a single term.

The comma is used in special conventions, as described below.

\(^\dagger\)This set of symbols has not been shown to be essential to the system. It serves more as a convenience to the coder and may introduce some undesirable flexibilities unless used with caution.
2.3 **The Basic Forms.**

The symbols defined above may appear in seven basic forms. Section 2.4 sets forth rules to be used in combining these basic forms to symbolize more complex logical conditions.

Let us define the quantity \( X \) as any single subscripted \( E, S, \) or \( T \). The basic forms may be written as:

1. \( X \) The single quantity. In this case \( X \) may also be a subscripted \( G \) or \( H \).

2. \( I_j X \) Execute \( X \) loopwise for the range of the subscript indicated in the definition of \( I_j \).

3. \( C_i X \) Execute \( X \) only if the condition \( C_i \) is satisfied. In this form, \( X \) may be the subscripted transfer symbol \( G_i \), resulting in a conditional transfer.

4. \( X C_i \) Execute \( X \), test \( C_i \); if \( C_i \) is not satisfied, execute \( X \) again, test \( C_i \) and continue iterating until \( C_i \) is satisfied.

5. \( C_j(X_a, X_b) \) If \( C_j \) is satisfied, do \( X_a \) and skip \( X_b \). If \( C_j \) is not satisfied, skip \( X_a \) and execute \( X_b \). \( X_a \) and/or \( X_b \) may be the subscripted symbol \( G_i \) in this form.

6. \( (X_a, X_b)C_j \) Execute \( X_b \), test \( C_j \). If \( C_j \) is not satisfied, do \( X_a \) and \( X_b \), test \( C_j \) and iterate until \( C_j \) is satisfied. In a control sense, this is an iteration loop in which the statement, \( X_a \), is executed in every iteration except the first.

7. \( I_j(X)C_k \) After each cycle of the \( I_jX \) loop, test \( C_k \) and if satisfied, leave the loop and calculate the next term in the control statement. If \( C_k \) is not satisfied, do the next cycle, test \( C_k \) again, etc. Exit from the \( I_jX \) loop is made as in form 2 unless \( C_k \) is satisfied earlier.

In each of the above forms involving \( I_j \), any other range statement could be used (\( N_k \), \( L_i \), etc.). Note that all characters must be subscripted.
2.4 Combinations.

More complex operations may be expressed by combinations of
the above forms in which the following rules are applied.

1. Any basic form may be combined with another by using the
"continuation symbol" or plus sign (+). Thus, $E_1 + S_k$
means "execute the equations in $E_1$ as defined, then perform
the sequence of operations defined by $S_k$;" 

2. Any basic form may be placed within another basic form (or
within itself) in the position occupied by the symbol X of
region 2.3. Thus $I_k(C_j E_i)$, form 3 within form 2, means
"for each cycle of the $I_k(...)$ loop, calculate $E_i$ only if $C_j$
is satisfied; otherwise start the next cycle." Or, $I_k(J_i E_k)$,
form 2 within itself, means "for each value of the subscript
whose range is defined by $I_k$, execute the loop defined by
$J_i E_k$." This is the familiar loop within a loop.

Note that the basic form which replaces the X is set
off by parentheses to indicate order of performance. Paren-
theses are not required in forms 5, 6, or 7 since the form
itself sets off the X's by parentheses and commas.

3. Any meaningful combination created by the above two rules
may also be inserted within a basic form in place of an X.
This "second-order" combination may be illustrated by the
term $C_i(C_j E_i + S_k, I_k E_n)$, forms 3 and 1 combined using
rule 1 and placed, with form 2, within form 5. The meaning
of this term can best be illustrated by the flow diagram

- -- -

\[ C_i \quad \text{yes} \quad C_j \quad \text{yes} \quad E_k \]

\[ S_k \]

\[ m=0 \rightarrow E_n \]

\[ m-1 \rightarrow m \]

\[ m=1 \quad \text{yes} \]

\[ \text{no} \]

where $I_k$ is defined, for convenience of illustration, as
$m=10(-1)2$ and $C_i$? asks the question, "Is the $C_i$ condition
satisfied?"

Additional compounding may be achieved by repeated use of the
above rules.

-5-
2.5 C-Statements.

Further substitution may be made in the combinations and basic forms by replacing a single C condition with a C-Statement. Since the C's are defined as binary conditions, a logically arranged set of C's can have the effect of a single C. For this, we use the second definition of the symbol (+) and say that \( C_i + C_j \) means \( C_i \text{ or } C_j \). Similarly, let us define \( C_i C_j \) as \( C_i \text{ and } C_j \).

Substituting an expression consisting only of C's into a basic form or combination allows us to express conditions such as \( (C_1 + C_2 C_3)E_1 \), meaning "if \( C_1 \) is satisfied or if \( C_2 \) and \( C_3 \) are both satisfied, execute \( E_1 \); otherwise skip \( E_1 \) and continue to the next term."

C-statements may also contain negated conditions by using the negation symbol (\( \bar{\cdot} \)). Thus \( (C_1 + C_2 C_3)\bar{\cdot} \) would be the negation of the total situation defined in the above example. More complex statements of this type may be defined by the basic rules of Boolean Algebra.

When substituting a C-statement into a form or combination, parentheses are needed to encompass the statement only if the resulting terms are OR-ed at some point. Thus \( C_3 C_4 + (C_2 + C_5) \) should be enclosed by parentheses as \( (C_3 C_4 + (C_2 + C_5)) \), but the term \( C_3 C_4 (C_2 + C_5) \) need not be.

To illustrate, re-define example 1 as:

\[
U = \sum_{i=1}^{10} x_i + \begin{cases} 
    a & \text{(if } \Sigma > 0, \Sigma \leq 3 \text{ and } x_1 \geq 4) \\
    b & \text{(if } \Sigma > 3 \text{ or } x_1 < 4) \\
    0 & \text{otherwise}.
\end{cases}
\]

We could completely define this as follows:

**Example 1a**

<table>
<thead>
<tr>
<th>Control</th>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ldots + E_1 + I_1 E_2 + C_1 C_2 C_3 \bar{\cdot} (E_4, (C_2 \bar{\cdot} + C_3)E_3) + \ldots )</td>
<td>( E_1 : U = 0 )</td>
</tr>
<tr>
<td>( I_1 : i = 1(1)10 )</td>
<td>( E_2 : U = U + x_1 )</td>
</tr>
<tr>
<td>( C_1 : \Sigma &gt; 0 )</td>
<td>( E_3 : U = U + b )</td>
</tr>
<tr>
<td>( C_2 : \Sigma \leq 3 )</td>
<td>( E_4 : U = U + a )</td>
</tr>
<tr>
<td>( C_3 : x_1 &lt; 4 )</td>
<td></td>
</tr>
</tbody>
</table>
2.6 Sample problem.

Let us now illustrate some of the preceding ideas with another example. Assume we wish to solve the Laplace equation for a 10 x 10 mesh with \( x = 1 \) on the boundaries and \( x = 0 \) in the interior. Then:

**Example 2.**

<table>
<thead>
<tr>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 : I_1((C_1 + C_2 + (C_3 \cdot C_4))(E_1, E_2)) )</td>
</tr>
<tr>
<td>( S_2 : E_3 + I_2(J_2'E_4) )</td>
</tr>
<tr>
<td>( S_3 : S_1 + S_2C_5 + E_5 )</td>
</tr>
<tr>
<td>( I_1 : i = 1(1)10 )</td>
</tr>
<tr>
<td>( I_2 : i = 2(1)9 )</td>
</tr>
<tr>
<td>( J_1 : j = 1(1)10 )</td>
</tr>
<tr>
<td>( J_2 : j = 2(1)9 )</td>
</tr>
<tr>
<td>( C_5 : C - 0.001 &lt; 0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_1 : x_{i,j} = 1 )</td>
</tr>
<tr>
<td>( E_2 : x_{i,j} = 0 )</td>
</tr>
<tr>
<td>( E_3 : C = 0 )</td>
</tr>
<tr>
<td>( E_4 : E = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4 )</td>
</tr>
<tr>
<td>( C = C + \sqrt{E - x_{i,j}} )</td>
</tr>
<tr>
<td>( x_{i,j} = E )</td>
</tr>
<tr>
<td>( E_5 : \text{STOP} )</td>
</tr>
</tbody>
</table>

\( S_1 \) sets up the mesh, \( S_2 C_5 \) represents the main calculation, and \( E_5 \) stops the problem after the iteration is complete. By defining the additional range statements:

\( I_3 : i = 1(9)10 \) and \( J_3 : j = 1(9)10 \)

we could set up the mesh using no condition statements whatever:

\( S_1 : I_3(J_1E_1) + I_2(J_1E_2) + I_2(J_2E_2) \) which would probably improve the set-up from a time standpoint.

Note that the use of \( C \) and \( E \) as program variables is not restricted, even though the same symbols are used for control.
3. Algae Programming for the 704.

3.1 Basic Assumptions and Restrictions.

Since Algae I currently uses the Fortran compiler as an intermediate step in its assembly, it is assumed that the programmer has a sound knowledge of basic Fortran concepts and restrictions. In addition, the following rules apply:

1. All subscripts on the logical symbols (E,S,T,G, etc.) must be numerical and greater than zero. Subscripts on non-control symbols are restricted only by the Fortran regulations.

2. Formula numbers can be assigned by the Algae programmer only to FORMAT statements, and must lie within the range $1 \leq FN \leq 99$. All other formula numbering will be done by the Algae code under one of two options selected by the programmer.

3. Because of restriction 2, FREQUENCY statements cannot be used in Algae I.

4. Comments can appear only at the beginning of the deck or within the defined E's.

5. Identification punched in columns 73-80 will be reproduced on Algae I listings only if the cards are read onto tape by the off-line equipment.

3.2 Coding Conventions and Specific Restrictions.

The Fortran coding form is used to prepare the problem for keypunching. Specific instructions for defining each character are given below with the restrictions governing the use of this character in Algae I. Unless otherwise defined, subscripts must lie in the range $1 \leq i \leq 999$. There may be no more than 500 E's, each having 70 cards or less.

Each $E_i$ is a list, in Fortran language, of the "Things to be done" by one flow diagram "box" or closed set of equations. None of the statements listed within an $E$ are tested by Algae I for accuracy. Rather, Algae I substitutes the entire set of equations into the sequence of Fortran statements whenever the specific $E$ is requested by a control statement.

E's may contain comments at any point (they may consist entirely of comments), input-output statements, and/or the two control statements SENSE LIGHT i and PAUSE.* These are

*The control statement STOP should not be used, since it results in a logic error in FORTRAN.
the only control statements listed in chapter 4 of the Fortran manual that should ever be used by the Algae programmer. E's cannot contain FORMAT, DIMENSION, or EQUIVALENCE statements, and must be less than 70 cards in length. Multiple reference may be made to defined E's, as described in section 2.2.

Page 14 illustrates the method of defining E's. Note that the definition of E_32 is terminated by the appearance of another identification symbol, E_16. E_16 is in turn terminated by the symbol C_3. Fortran equations within an E may be continued from one card to the next as in normal Fortran notation.

There are two types of conditions in the Algae I program: Type I, illustrated by C_3 and C_5 on page 14 defines the test of a numerical condition. The term or expression within parentheses is compared to zero in a manner designated by the symbol appearing at the right of the parenthesis. If we denote the contents of the parenthesis by X,

(X) P asks, "Is X positive?"  (X > 0?)
(X) N asks, "Is X negative?"  (X < 0?)
(X) = asks, "Is X equal to zero?"  (X = 0?).

The symbols may be negated to obtain the remaining inequalities:

(X) P*, "Is X ≤ 0?"
(X) N*, "Is X ≥ 0?"
(X) *, "Is X ≠ 0?"

In Algae I, the numerical condition (including parentheses) must occupy at most 48 card columns.

Type II conditions, illustrated by C_1, C_2, and C_4 on page 14, define the test of a trigger, sense switch, or sense light. It is distinguished from a type I condition by the absence of a left parenthesis as the initial non-blank character. A comparator is also used to indicate whether the condition is satisfied when the trigger is on or off.
Typical type II conditions are:

| SENSE LIGHT 3 ON | S L 3 ON |
| SENSE SWITCH 6 OFF | SW 6 OFF |
| ACCUMULATOR OVERFLOW ON | A ON |
| DIVIDE CHECK ON | D ON |
| QUOTIENT OVERFLOW OFF | Q OFF |

For the benefit of programmers who are in a hurry or dislike spelling, Algaa I will accept the abbreviations at the right in the above listing. An S, A, D, or Q must appear in the first non-blank position to avoid an error, and the ON/OFF condition must be specified. Do not attempt to abbreviate the SENSE LIGHT i instruction when it is used within an E to turn a specific light on.

A $C_i$ of the two above types must have a numerical subscript within the range $1 \leq i \leq 149$.

C-statements, illustrated by $C_{153}$ on page 14 and defined in section 1.5, enable the programmer to use repeatedly a complex condition with a minimum of writing. A C-Statement must have as its identification a numerically subscripted $C_i$ with $150 \leq i \leq 200$. In Algaa I, a C-Statement cannot contain another C-Statement, and must be complete on one card (no continuations).

A C Group has been defined as any closed grouping of conditions, including condition statements. Thus if a statement contains the expression $... \times C_4 + C_1(C_2 + C_3) \times C_{155} E_2 + ...$, $C_4$ and $C_1(C_2 + C_3) \times C_{155}$ are C groups. The maximum number of C's which may appear in a C group is 70. In the second group above, this would include all C's within the statement $C_{155}$ but not $C_{155}$ itself.

$I_1 - N_1$: Range statements (loop conditions) are identified by the numerically subscribed characters $I, J, K, L, M, N$. Examples of range statements are $I_4$ and $M_2$ on page 14. All range statements have the form $a=b(c)d$ where:

- $a$ is the subscript whose range is being defined,
- $b$ is the first value to be taken by $a$,
- $c$ is the increment of change, and
- $d$ is the final bound on $a$. The final value of $a$

is $\leq d$ if $c > 0$, or $\geq d$ if $c < 0$. 

-9-
The following rules apply:

1. \( a \) may consist of any four Hollerith characters, the first of which is I,J,K,L,M, or N.

2. \( b \) and \( d \) may be any non-zero symbols recognized by Fortran as fixed point constants or variables. They can never assume negative values. Note that Fortran prevents the use of subscripted fixed point variables for \( b \) and \( d \) in DO's.

3. \( c \) may be any non-zero symbol (positive or negative) recognized by Fortran as a fixed point constant or variable. If \( d \) is greater than \( b \), \( c \) must be greater than zero. If \( b \) is greater than \( d \), Algae I will set up a reverse DO loop using a dummy variable. In this case, \( c \) must be preceded by a minus sign to indicate the reversal of the loop condition. The dummy variable is formed by repeating the first letter of \( a \) three times and adding the remainder of the term (see below). Care should be taken to avoid using this variable in the problem.

Some acceptable forms would be:

\[
\begin{align*}
IVAL &= \text{IMAX}(-\text{IDELTA})\text{MIN} & \text{Dummy Variable: } & \text{IIIIVAL} \\
J32 &= 45 (-3) \text{JLOW} & \text{Dummy Variable: } & \text{JJJ32} \\
M3K2 &= 1 (\text{INCR}) 10 \\
I &= \text{II (3) 15}.
\end{align*}
\]

It is not necessary for the symbol \( a \) to be the same as the identification symbol. For example, in \( I_i \) on page 14 we define the range of the subscript \( JT \). The subscript on the identification must lie in the range \( 1 \leq i \leq 999 \). Only 150 range statements can be defined in a problem. Range statements, once defined, may be used repeatedly.

All 4 fields (\( a \), \( b \), \( c \), and \( d \)) must be filled. Algae does not make the assumption that a missing term is equal to 1.

\( S_i \) and \( T_i \). These control statements are written for the 704 in the same manner as those defined in examples 1, 1a, and 2 (see page 14 for further illustrations). Note that all subscripts are numerical and non-zero. Subscripts may be written as
smaller characters, but caution should be taken to keep them above the line to avoid keypunching errors. Care should also be taken to make commas distinctively different from the subscript i.

S's and T's may be continued over several cards without the use of a continuation symbol in column 6, however a statement must have less than 1000 characters (not counting subscripts) and must have any subscripts completed on one line. One problem may contain a maximum of 450 S's and T's combined.

Subscripts on S_i must lie in the range 1 ≤ i ≤ 999 but subscripts on T_j are restricted to the range 1 ≤ j ≤ 100. To provide the multiple exit to a T sequence, the dummy variable LLxxx is formed, where xxx is the subscript on the defined T. Thus, T_{56} would produce a dummy variable LL56 which should not be used elsewhere in the code.

G_i and H_i. These characters need never be defined and should appear only in S or T statements. In both cases the subscript i should lie between 1 ≤ i ≤ 200. Violation of this restriction will not produce an error stop, but might lead to a swifter depletion of the allowed quantity of formula numbers.
3.3 Deck Ordering and Operation.

1. Deck ordering.

A problem designed for Algae I assembly may be considered to be divided into two sections. Section I must contain all

1. Control cards (see part 2 below)
2. Initial comments (all other comments must appear within E's)
3. Format statements
4. Dimension statements
5. Equivalence statements.

These cards may be arranged in any order in section I. All cards in section I must be loaded before the first card of section II. Section II consists of all the defined E's, S's, T's, C's and range statements, in any order.

2. Sense switches.

Sense switches 5 and 6 are used by the 8K ALGAE I. They have the following effect:

Sense switch 5:
If this switch is down, and if errors have been detected by the compiler, the FORTRAN code, if any, which has been produced will be listed on tape 9.

Sense switch 6:
If this switch is up, each formula in an E will be assigned a formula number. If the switch is down, formula numbers will be assigned only where necessary to define control of the problem.

3. Preparation of Algae System Tape.

The Algae System Tape is tape # 6. Because of the large number of cards involved, Algae should not be read by the card reader unless the installation does not possess enough tape units to permit the entire system to function from tapes. To write the system tape, place a blank tape on logical tape 6 and load the ALGAE 8K deck in the card reader. Press "LOAD CARDS." Note that it is not necessary to clear the machine or reset the console.

**WARNING:** DO NOT ATTEMPT TO WRITE TWO TAPES SIMULTANEOUSLY BY TURNING THEM BOTH TO 6. THE WRITING PROGRAM READS THE TAPE IT HAS WRITTEN, AND COMPARES IT WITH THE CONTENTS OF MEMORY. The 704 cannot read 2 tapes at once, and the program will continue to try to write a good tape indefinitely.
4. Operation

Input may be from cards or from a tape prepared on peripheral card-to-tape equipment. Each problem must be loaded separately, whether it is loaded by cards or by tape, as the input tape is destroyed in the process of compiling.

Machine set-up is as follows:

a) Tapes by logical tape number.
   1. FORTRAN
   2. blank
   3. blank or BCD input (see above)
   4. blank
   5. blank
   6. ALGAE
   9. tape-to-printer BCD tape

b) Printer. Share # 2 board.

c) Punch. Blank cards.

d) Card reader.
   1. ALGAE RREC 1 card.
   2. Problem decimal deck, if input is not on tape 3. Otherwise, no cards.
   3. No cards.

To initiate compiling, press the "LOAD CARDS" button. When the deck has been completely read in, and the heading printed on the printer, place ALGAE RREC 2 in the card reader and run it in to ready. If it is desired, the next deck to be compiled may follow RREC 2 in the card reader.

There are three program stops.

1. If a card being read in has a non-Hollerith character, the program will stop with $120_8$ in the SR. The card which has just been read should be corrected, it and all unread cards replace in the card reader, and the "START" button pushed.

2. If compiling halts because of an error, a stop will occur with $5$ in the address of the SR. Press "START" to load a self-loading card.

3. If there has been a machine error, there will be a stop with $1747_8$ in the SR. Do not proceed until the trouble has been determined.
Note: The RREC 2 card brings into memory a code which transfers the FORTRAN output from tape 2 to tape 9. If the local FORTRAN has such a routine, do not use RREC 2.

Note: As soon as RREC 2 has been read, a new tape 3 with new BCD input may be hung.
**Fortran Statement**

**Examples of Algol I Definitions:**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Comments</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E32</td>
<td>Comments may appear anywhere within an E</td>
<td>May be</td>
</tr>
<tr>
<td></td>
<td>$x = x * 2 + 3.2 * y + \text{SORTF(VM)}$</td>
<td>punched</td>
</tr>
<tr>
<td></td>
<td>$T(i) = u(npos) * 2 + v(nmin) + w(n3) - \text{SORTF}(x * 3) / 4.0 * n1$</td>
<td>here -</td>
</tr>
<tr>
<td>E16</td>
<td>pause</td>
<td>printed</td>
</tr>
<tr>
<td>C</td>
<td>Be sure to identify each comment by a column *1.</td>
<td>By Algol</td>
</tr>
<tr>
<td>C3</td>
<td>$(x * 2 - 3.0 * y(i) + 7.29) * p^2$</td>
<td>*Algo</td>
</tr>
<tr>
<td>C5</td>
<td>$(y - 2) = \text{divide} \check \phi n$</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>sense * light * 3 * \phi f</td>
<td>|</td>
</tr>
<tr>
<td>C2</td>
<td>$s * s w * 4 * \phi n$</td>
<td>|</td>
</tr>
<tr>
<td>C153</td>
<td>$(c_1 + (c_2 c_4 \ast + c_5) c_6 \ast$</td>
<td>|</td>
</tr>
<tr>
<td>T4</td>
<td>$j t = 3(15) \ast$</td>
<td>|</td>
</tr>
<tr>
<td>M2</td>
<td>$m3 = \max(-3) \min$</td>
<td>|</td>
</tr>
<tr>
<td>S4</td>
<td>$c_1 (E_3 + E_4 c_6 + I_4 (M_2 (E_6 + I_3 E_4)) + T_3 + C_1 (S_5) S_6) + J_3 (E_4 + I_2 (C_1 C_6 E_7 + S_3))) + E_16$</td>
<td>|</td>
</tr>
<tr>
<td>T74</td>
<td>$k_2 (E_1 + (E_2 \ast E_3) c_4) + C_1 \varpi$</td>
<td>|</td>
</tr>
</tbody>
</table>

(The above statements are not intended to form a consistent set.)
3.4 Sample Problems

On the following pages will be found two sample problems coded in Algae. The first problem is correctly coded. Most questions arising about the use of forms, control cards, etc., may be answered by careful perusal of this listing.

In the second code, "Laplace", on page 25 ff., intentional errors have been made so that the reader may become familiar with the diagnostic procedures used in Algae I. For a more complete discussion of diagnostics, see section 4. of this writeup.
ALG C ALGAE DEMONSTRATION CODE
C ARITHMETIC-GEOMETRIC MEAN COMPUTER
DIMENSION XP(25), YP(25), X(25), Y(25), FX(25)
EQUIVALENCE (X, XP); (Y, YP)
1 FORMAT (2I3)
2 FORMAT (35 H1) ALGAE DEMONSTRATION CODE / 43 H1
A C ARITHMETIC-GEOMETRIC MEAN COMPUTER / 8 HC 1
B= 13, 4 H1; J= 13
3 FORMAT (36 HO, X(I), Y(J), F(I,J))
4 FORMAT (3 E13.4)
5 FORMAT (36 HO, F(I,J) FROM PREVIOUS ITERATION / 8
AE 13.41)
6 FORMAT (35 HO, SUMX, SUMY, AGH / 3E13.4)
CE1 COMPUTE 2 P1, SET UP DP, DYP,
CAPI = ICAP
CAPJ = JCAP
TWOP = 2.0 * 3.14159
DX = TWOP / CAPI
DY = TWOP / CAPJ
E23 T = T-1
C COMPUTE THE VALUES OF X PRIME,
XP(I) = T * DX
E24 T = T-1
C COMPUTE THE VALUES OF Y PRIME,
YP(J) = T * DY
E3 AM = AM1
GM = GM1
E4 X(I) = SINF (XP(I))
C COMPUTATION OF Xa
CE5 FIRST COMPUTATION OF Ya
Y(J) = COSF (EXPF (YP(J)))
CE6 SECOND COMPUTATION OF Ya
Y(J) = COSF (EXPF (-YP(J)))
E7 AM = X(I)
C PREPARE AGM COMPUTATION.
GM = Y(J)
CE8 AGM COMPUTATION.
C ARITHMETIC MEAN.
AM1 = (AM + GM) / 2x0
C GEOMETRIC MEAN.
GM1 = SORIF (AM * GM)
E2 AGM = AM1
E9 FXY(1,J) = AGM
E10 FXY(I,J) = 0.0
E11 FXY(I,J) = -1.0
E12 FXY(I,J) = 1.0
CE13 INITIALIZE SUMS.
SUMX = 0.0
SUMY = 0.0
E14 SUMX = SUMX + ABSF (XP(I))
E22 PRINT 50; SUMX, SUMY, AGH
E15 SUMY = SUMY + ABSF (YP(J))
CE16 SET TO COMPUTE AGM OF SUMS.
AM = SUMX
GM = SUMY
SSENSE LIGHT 1
F18 READ 1, ICAP, JCAP
PRINT 2; ICAP, JCAP
F19 PRINT 4; X(I), Y(J), FXY(I,J)
E20 PRINT 3
F21 PRINT 5, (FXY(I,J), I = 1, ICAP), J = 1, JCAP
F25 SSENSE LIGHT 2
CE26 IGNORE TEST.
CE26
N1  I = ICAP (*1) 1
M7  J = 1 (1) JCAP
C1  (X(I)) =
C2  (Y(J)) =
C3  (X(I)) N
C4  (Y(J)) N*
C5  SWITCH 6 OFF
C7  ACCUM OVERFLOW ON
C151 (C1 C4 C2* + C2 C3* C1*)
C8  SIG LITE 1 OFF
C9  (ABS (AM1 - GM1) / GM1 = 00001) N
C10 SENSE LT 2 ON
T1  (E3* E8) C9 + E2
T2  E1 + N1 E23 + M2 E24
S1  C3* C1* C4 C2* (E7 + T1 + C7 (E10* E9) + C151 (E12* C3 C4*)
    (E11* E10*)
S2  T2 + N1 E4 + C8 (M2 E5* M2 E6) + C5* E20
    +N1 (M2 (S1 + C5* E19)) + E21
S3  C7 E26 + H1 + E18 + H2 + S2 + C10* (S4* G1)
S4  E25 + T2 + E13 + N1 E14 + M2 E15 + E16 + T1 + E22 + G2
ALGAE OBJECT CODE FROM FORTRAN TAPE 2

C ALGAE DEMONSTRATION CODE
C ARITHMETIC-GEOMETRIC MEAN COMPUTER
C
DIMENSION XP(25), YP(25), X(I), Y(I), F(J,I)

EQUIVALENCE (X, XP(I), Y, YP(I))

1 FORMAT (2I3)
2 FORMAT (35 H)
3 FORMAT (36 H)
4 FORMAT (3 E 13.4)
5 FORMAT (36 H)
6 FORMAT (35 H)

C ALGAE DEMONSTRATION CODE / 45 H
A ARITHMETIC-GEOMETRIC MEAN COMPUTER / 8 HO I
B = 13, 4 H J = 13)
C FORMAT (36 HO, X(I)) Y(J) F(I,J)
D FORMAT (3 E 13.4)
E FORMAT (36 HO, F(I,J) FROM PREVIOUS ITERATION / (B
AE 13.4))
F FORMAT (35 HO SUMX SUMY AGM / 3E13.4)
G IF ACCUMULATOR OVERFLOW 0144, 0145 C007
H
C CONTINUE
I IGNORE TEST.,
C
0112 CONTINUE
READ 1, ICR, JCR P018,01
C
PRINT 2, ICR, JCR P018,02
C
0113 CONTINUE
LL002#0002 T002
GO TO 0111
C
0132 CONTINUE
D00133 III = 1, ICR, 1
1 M0011
I = 1 + ICR-1111
N001
X(I) = SINF (XP(I)) E004,01
C COMPUTATION OF X.
C
0133 CONTINUE
IF (SENSE LIGHT 1) 0135, 0134 C008
0134 CONTINUE
D00129 J = 1, ICR, 1
C
FIRST COMPUTATION OF Y,
Y(J) = COSF (EXP (YP(J))) C006
E005
C
0129 CONTINUE
GO TO 0136
C
0135 CONTINUE
D00130 J = 1, ICR, 1
C
SECOND COMPUTATION OF Y,
Y(J) = COSF (EXP (-YP(J))) E006,01
C
0130 CONTINUE
IF (SENSE SWITCH 6) 0137, 0138 C005
C
0137 CONTINUE
PRINT 3 E020,01
C
0138 CONTINUE
D00139 III = 1, ICR, 1
1 M0011
I = 1 + ICR-1111
N001
C
D00131 J = 1, ICR, 1
C
AM = X(I) 0125, 0102, 012 C003
I IF(X(I))
0102 IF(X(I)) 0125, 0102, 012 C003
0101 IF(Y(J)) 0125, 0102, 012 C003
0100 IF(Y(J)) 0124, 0125, 0124 C002
C
0124 CONTINUE
AM = X(I)
C PREPARE AGM COMPUTATION.
C
GM = Y(J)
LL001#0002 T001
GO TO 0110
C
0117 CONTINUE
IF ACCUMULATOR OVERFLOW 0118, 0119 C007
C
0118 CONTINUE
FXY(I,J) = 0.0
GO TO 0120
0119 CONTINUE
FXY(I,J) = AGM
0120 CONTINUE
GO TO 0126
0125 CONTINUE
IF(X(I))
0107 IF(Y(J))
0106 IF(Y(J))
0105 IF(Y(J))
0104 IF(X(I))
0103 IF(X(I))
0121 CONTINUE
FXY(I,J) = 1.0
GO TO 0129
0122 CONTINUE
IF(X(I))
0108 IF(Y(J))
0114 CONTINUE
FXY(I,J) = -1.0
GO TO 0116
0115 CONTINUE
FXY(I,J) = 0.0
0116 CONTINUE
0123 CONTINUE
IF (SENSE SWITCH 6)
0127 CONTINUE
PRINT 4, X(I), Y(J), FXY(I,J)
0128 CONTINUE
0131 CONTINUE
0139 CONTINUE
PRINT 5, (FXY(I,J)), I = 1, ICAP, J = 1, JCAP)
IF (SENSE LIGHT 2)
0146 CONTINUE
SENSE LIGHT 2
LL002#0001
GO TO 0111
0140 CONTINUE
C INITIALIZE SUMS:
SUMX = 0.0
SUMY = 0.0
DO0141
III = 1, ICAP, 1
SUMX = SUMX + ABSF (XP(I))
DO0142
J = 1, JCAP, 1
SUMY = SUMY + ABSF (YP(J))
0141 CONTINUE
0142 CONTINUE
C SET TO COMPUTE AGM OF SUMS.
AM = SUMX
GM = SUMY
SENSE LIGHT 1
LL001#0001
GO TO 0110
0143 CONTINUE
PRINT 6, SUMX, SUMY, AGM
GO TO 0113
0147 CONTINUE
GO TO 0112
0111 CONTINUE
C COMPUTE 2 PT, SET UP DXP, DYP.

CAPI = ICAP
CAPJ = JCAP
TWOPI = 2*0.5*14159
DX = TWOPI / CAPI
DY = TWOPI / CAPJ
DO0149 I = 1, ICAP, 1
   N001 I = 1 + ICAP-1
   C COMPUTE THE VALUES OF X PRIME,
   XPI(I) = T * DX
   0149 CONTINUE
   DO0150 J = 1, JCAP, 1
   T = J-1
   C COMPUTE THE VALUES OF Y PRIME,
   YPI(J) = T * DY
   0150 CONTINUE
   GO TO0140 #0132, L002
   0110 CONTINUE
   GO TO 0151
   0152 CONTINUE
   AM = AM1
   GM = GM1
   C AGM COMPUTATION
   C ARITHMETIC MEAN
   AM1 = (AM + GM) / 2.0
   C GEOMETRIC MEAN
   GM1 = SRTF (AM * GM)
   IF (ABS (AM1 - GM1) / GM1, 0001)
   0153 CONTINUE
   AGM = AM1
   GO TO0143 #0117, L001
### STORAGE FOR VARIABLES APPEARING IN DIMENSION OR EQUIVALENCE SENTENCES

<table>
<thead>
<tr>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FY 32717 77715</td>
<td>XP 32767 77777</td>
<td>X 32767 77777</td>
<td>YP 32742 77746</td>
<td>Y 32742 77746</td>
<td></td>
</tr>
</tbody>
</table>

### STORAGE FOR VARIABLES WHICH DO NOT APPEAR IN DIMENSION OR EQUIVALENCE SENTENCES

<table>
<thead>
<tr>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
<th>DEC OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 32092 76534</td>
<td>111 32091 76533</td>
<td>ICAP 32090 76532</td>
<td>GM 32089 76531</td>
<td>GH 32088 76530</td>
<td></td>
</tr>
<tr>
<td>DY 32087 76527</td>
<td>DX 32086 76526</td>
<td>CAPJ 32085 76525</td>
<td>CAPI 32084 76524</td>
<td>AM 32083 76523</td>
<td></td>
</tr>
<tr>
<td>AM1 32082 76522</td>
<td>AGM 32081 76521</td>
<td>JCAP 32080 76520</td>
<td>J 32079 76517</td>
<td>LL001 32078 76516</td>
<td></td>
</tr>
<tr>
<td>LL002 32077 76515</td>
<td>SUMX 32076 76514</td>
<td>SUMY 32075 76513</td>
<td>T 32074 76512</td>
<td>TMOPI 32073 76511</td>
<td></td>
</tr>
</tbody>
</table>

### EXTERNAL FORMULA NUMBERS WITH CORRESPONDING INTERNAL FORMULA NUMBERS AND OCTAL LOCATIONS

<table>
<thead>
<tr>
<th>EFN</th>
<th>IFN</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>LOC</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>00000</td>
</tr>
<tr>
<td>132</td>
<td>22</td>
<td>00044</td>
</tr>
<tr>
<td>130</td>
<td>36</td>
<td>00117</td>
</tr>
<tr>
<td>101</td>
<td>48</td>
<td>00166</td>
</tr>
<tr>
<td>119</td>
<td>60</td>
<td>00216</td>
</tr>
<tr>
<td>105</td>
<td>68</td>
<td>00241</td>
</tr>
<tr>
<td>108</td>
<td>76</td>
<td>00265</td>
</tr>
<tr>
<td>126</td>
<td>84</td>
<td>00277</td>
</tr>
<tr>
<td>146</td>
<td>102</td>
<td>00374</td>
</tr>
<tr>
<td>147</td>
<td>126</td>
<td>00474</td>
</tr>
<tr>
<td>152</td>
<td>146</td>
<td>00577</td>
</tr>
</tbody>
</table>

### SUBROUTINES OBTAINED FROM LIBRARY

<table>
<thead>
<tr>
<th>(DBC)</th>
<th>491 00753</th>
<th>(CSH)</th>
<th>1108 01760</th>
<th>(BDC)</th>
<th>1069 02101</th>
<th>(FIL)</th>
<th>1119 02137</th>
<th>(LEV)</th>
<th>1132 02154</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RTN)</td>
<td>1159 02207</td>
<td>(SPH)</td>
<td>1639 03147</td>
<td>SGRT</td>
<td>1731 03303</td>
<td>COGS</td>
<td>1753 03331</td>
<td>SIN</td>
<td>1796 03334</td>
</tr>
<tr>
<td>EXP 1810 03422</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ALGAE LAPLACE EQUATION ROUTINE -- TYPE 1 MESH

1. FORMAT (30H, ALGAE LAPLACE EQUATION SOLVER/ 49H 10 X 10 MESH/
   A WITH BOUNDARIES SET TO 1)
2. FORMAT (10 E 12.4)
3. FORMAT (36H ITERATION ON LAPLACE EQUATION MESH / (10E11.3)

C THIS COMPILING WILL DEMONSTRATE ERROR FORMATS.
C
S2 E3 + 12 (J2 E)
S1 I1 1 J1 ((C1 + C2 + (C3 C4)* + (E1, E2))
S3 S1 + (S2 C6 E7) C5 + E5 + E8
I1 I = 1 (1) 12
I2 J = 2 (1) 9
J1 J = 1 (1) 10, AND SO ON
J2 J = 2 (1) 9
C1 (I=I)
C2 (J=J0) E =
C3 (J+1) N
C4 (J-9) P
C5 C = 0.0011 N
C6 (SW 6 ON)
CE1 SET UP BOUNDARY VALUES OF MESH
  X(I+J) = 1.0
CE2 SET INTERIOR TO ZERO
  X(I+J) = 0.0
E3 C = 0.0
CE4 CALCULATE DIFFERENCES,
  E = (X(I,J+1) + X(I+1,J) + X(I,J+1) + (X(I,J)+1)) / 4.0
C SUM DIFFERENCES,
  C = C + ABSF (E - X(I,J))
C AND SET NEW VALUE OF X(I,J+1)
  X(I+J) = E
E5 PRINT 1
  PRINT 2* (X)
E6 PAUSE
E7 PRINT 3* (X)
ALGAE LAPLACE EQUATION ROUTINE -- TYPE 1 MESH

THIS CODE SOLVES THE LAPLACE EQUATION ON A 10 X 10 MESH

WITH BOUNDARIES SET TO 0.

DIMENSION X(10, 10)

1 FORMAT (10H) ALGAE LAPLACE EQUATION SOLVER/ 43H 10 X 10 MESH

2 FORMAT (10 E 12.4)

3 FORMAT (136HO ITERATION ON LAPLACE EQUATION MESH /* (10E11.3)

THIS COMPILING WILL DEMONSTRATE ERROR FORMATS.

0107 CONTINUE
0104 CONTINUE
0102 C = 0.0

000103 J = 2, 9, 1

000101 J = 2, 9, 1

0101 CONTINUE
0103 CONTINUE

IF (SW 6 ON)

0105 CONTINUE
0106 PRINT 3, (X)

CODING ERROR

0108 CONTINUE
0109 PRINT 1

0110 PRINT 2, (X)

HALT

FIX AND RE-TRY

0011 DIAGNOSTICS

CODE 07563
4. Error Procedure and Description of Diagnostics

Algae I is designed to detect errors in source programs presented to it, and to give as part of its output as much information as is possible concerning those errors.

The compiling process may be divided into two phases. In Phase 1, the source program is read into the 704 DPM, statements are checked for duplication and improper subscripts, and diverse tables needed in Phase 2 are set up. In Phase 2 the FORTRAN source program is compiled. Detection of an error in Phase 1 prevents the execution of Phase 2. Detection of an error in Phase 2 prevents the execution of FORTRAN.

Errors may be divided into four major classes. These are:

I. Machine errors.
II. Phase 1 source program errors.
III. Phase 2 source program errors which are relatively limited in their effect upon the code.
IV. Phase 2 source program errors whose nature makes further compiling impossible.

Class I errors may cause a program stop at 17478. An identification number, ALPHA, is printed on the line printer together with a brief description of the failure. Class II, III, and IV errors cause a statement to be printed on the off-line listing, containing an identification number, ALPHA, and a brief description of the source program error. Class II errors will cause the next problem to be loaded at the end of Phase 1. Class III errors will cause the next problem to be loaded at the end of Phase 2. Class IV errors will cause the next problem to be loaded whenever they are detected. Before leaving a problem the total number of source program errors which have been detected is printed both on the line printer and off-line listing.

Information printed about errors is listed in the following format: H ID COMMENT CHAR TYPE ALPHA
The H position contains the word HALT if processing of the problem is interrupted for any reason. ID is the identification of the objectionable statement when available. COMMENT is a brief description of the error. CHAR is the term of the statement, if any, to which exception is taken. TYPE indicates whether the error

-27-
is a machine failure or source program mistake. ALPHA is the octal location at which the error was detected. By looking up this number in the following ALPHA table, a slightly expanded description may be obtained.

There are two symbols which may occur in the CHAR column on the listing which are the result of coding errors, but which do not appear in the source program at any point.

The first of these is $W_0$. This character is inserted at one stage in the code when a character has been determined to be improper in some respect. The insertion of $W_0$ permits Algae I to continue compiling, and also permits the programmer to trace the effect of his errors on the resulting code.

The second is an $E$ with subscript $\geq 1000$. This symbol is the identification of the entire coded contents of a parenthetical expression. It sometimes happens that a program will give rise to a vacuous parenthesis level, or other similar error. Then, when it is desired to incorporate this grouping into the rest of the code, the $E_{1000}$ symbol is printed.

The format of the ALPHA table is as follows:

<table>
<thead>
<tr>
<th>ALPHA</th>
<th>COMMENT</th>
<th>CLASS of error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature of error</td>
<td>What to do about it</td>
<td></td>
</tr>
</tbody>
</table>

ALL ALPHA's are in octal.
<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Subscript Range</th>
<th>max. no. defined</th>
<th>no. of times each may be used</th>
<th>no. of cards no. of char.</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_i$</td>
<td>$1 \leq i \leq 999$</td>
<td>500</td>
<td>1</td>
<td>70 / 1000</td>
<td>Control statement</td>
</tr>
<tr>
<td>$T_i$</td>
<td>$1 \leq i \leq 100$</td>
<td>100</td>
<td>100</td>
<td>70 / 1000</td>
<td>Subroutine</td>
</tr>
<tr>
<td>$C_i$</td>
<td>$1 \leq i \leq 149$</td>
<td>149</td>
<td>Indefinite</td>
<td>$\frac{1}{4}$ including blanks &amp; ()</td>
<td>Simple Condition</td>
</tr>
<tr>
<td>$C_i$</td>
<td>$150 \leq i \leq 200$</td>
<td>51</td>
<td>Indefinite</td>
<td>$\frac{1}{66}$</td>
<td>C-statement. May not contain another C-statement.</td>
</tr>
<tr>
<td>$I_i - N_i$</td>
<td>$1 \leq i \leq 999$</td>
<td>150</td>
<td>Unlimited</td>
<td>$\frac{1}{10}$</td>
<td>Range statement</td>
</tr>
<tr>
<td>$G_i$</td>
<td>$1 \leq i \leq 200$</td>
<td>200$^+$</td>
<td>Unlimited</td>
<td>0</td>
<td>Transfer statement</td>
</tr>
<tr>
<td>$H_i$</td>
<td>$1 \leq i \leq 200$</td>
<td>200$^+$</td>
<td>1</td>
<td>0</td>
<td>Point of transfer entry.</td>
</tr>
</tbody>
</table>

$^+$ may be greater than 200, but in large problems the available formula numbers may be exhausted.