SP-2214/000/00

SELF-INSTRUCTIONAL JOVIAL MANUAL

CHAPTERS 1, 2, 3 & 4

Donald I. Cutler

15 October 1965

SP-2214/000/00

SP *a professional paper*

SELF-INSTRUCTIONAL JOVIAL MANUAL

CHAPTERS 1, 2, 3 & 4

by

Donald I. Cutler

SYSTEM DEVELOPMENT CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA

15 October 1965

BLANK PAGE

INTRODUCTION

This document has been written to be used as a self-instructional JOVIAL training manual. The format used in a self-instructional manual is quite different from that of a reference manual for experienced programmers. Hence, this is not to be considered as a reference manual for experienced programmers.

The manual presupposes no computer programming experience on the part of the student. If a student has had previous programming experience, this should enable him to cover the material more rapidly. This is one of the advantages of a self-instructional manual.

Another advantage is that students can begin to learn at once when their assignment to the programming vocation has been made. They do not need to wait until a sufficient number of people have been assigned to justify a formal class.

Thus the two main advantages of a self-instructional manual are:

1) Sensitivity to individual differences due to varying backgrounds and abilities.

2) Independence from a formal class.

CHAPTER 1

INTRODUCTION TO JOVIAL

JOVIAL is a language with which one can write programs to solve problems on a computer. A program is a sequence of instructions to a computer. A program contains the logic which is designed to solve a particular problem.

Each type of computer is electrically designed to be sensitive to a specific set of computer instructions. These are usually coded in the memory of the computer in what is called the binary language. This is a language consisting of combinations of 0's and 1's. The binary language is called a low level language.

Usually each type of computer has a set of symbolic instructions corresponding to the set of binary instructions. These are coded using alphabetic abbreviations and are called symbolic machine-instructions. This symbolic machine language is called an intermediate level language.

A disadvantage of symbolic machine language is that it is different for each make of computer. Furthermore, a programmer using symbolic machine language has to pay close attention to details peculiar to his machine. This detracts from his efforts to form the logic connected with the solution of the problem he is programming.

To overcome the aforementioned disadvantages of intermediate level languages, high level languages have been developed.

A high level programming language is one which more closely approaches the language of English and mathematics. It permits the programmer to be less concerned about the individual peculiarities of the specific computer involved and enables him to concentrate more conveniently on the logic involved in the solution of a problem.

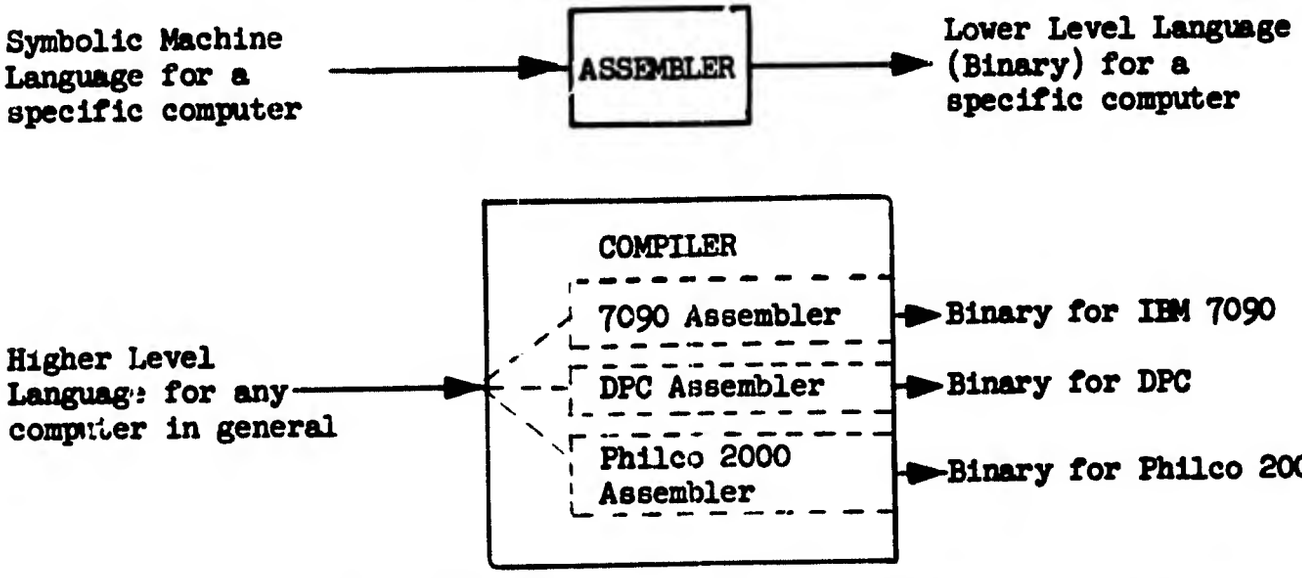
JOVIAL is a high level language. Other higher level languages are ALGOL, MAD, FORTRAN, COBOL, NELIAC etc. JOVIAL was developed by the System Development Corporation to be used for programming large scale command and control systems. Work on the language began in 1959. Since that time, many modifications and improvements have been made to the language.

The name JOVIAL is an acronym which is derived from Jules Own Version of the International Algebraic Language. Jules Schwartz of the System Development Corporation was the scientist in charge of the initial language development. Several colleagues of his supplied the name one time when he was away on a business trip and the name stuck. JOVIAL it is.

A computer program written in symbolic machine language must be translated to binary since the computer operates using instructions coded in binary. A computer program, called an assembler, is used to perform the translation from the intermediate level language to the lower level language.

Similarly a computer program written in a higher level language must be translated to the binary language of the specific machine involved. The computer program which performs this translation is called a compiler.

JOVIAL compilers are in existence for several makes of computers, two of the most well-known being the IBM 7090 computer, and the AN/FSQ-31V or DPC (Data Processing Central).



The above diagram is representative of the functions of assemblers and compilers. Usually a compiler contains only one assembler. For example, a JOVIAL compiler which translates JOVIAL program statements to IBM 7090 binary instructions, would contain in it only a FAP assembler (the 7090 assembler).

It is probably apparent that functions in addition to assembly are performed by a compiler. This is because the compiler has to translate the higher level language into the symbolic machine language of the assembler involved. Also each statement or instruction in a higher level language is often translated into many symbolic machine instructions.

The above diagram also illustrates that a program written in a higher level language can be operated on different computers by having the translation to binary performed using the appropriate compiler. This is a distinct advantage over writing at the intermediate language level, where the program can be operated only on the computer for which it was written.

EXERCISES

1.1 a. What are the names of the three levels of programming languages?

_____

_____

_____

b. To which level does the JOVIAL programming language belong?

_____

c. What is the general name of a program used to translate programs written in an intermediate level language to one of a lower level language?

_____

d. What is the general name of a program used to translate programs written in a higher level language to one of a lower level language?

_____

e. Can a program written in a higher level language be operated on more than one make of computer?_____

If yes, how?_____

_____

ANSWERS TO PRECEDING EXERCISES

1.1 a. The three levels of programming language are:

low level language

intermediate level language

high level language

b. JOVIAL is a high level programming language.

c. Assembler

d. Compiler

e. Yes, by having the program translated by the appropriate compiler.

CHAPTER 2

ITEMS, THE ASSIGNMENT STATEMENT, AND THE 4 BASIC ARITHMETIC OPERATIONS

Programs in JOVIAL perform essentially two functions. They manipulate data and they make decisions based upon the value of the data.

To manipulate the data conveniently, the data is contained in variables called items. This is quite analogous to mathematics where we might have the equation A = B + 6.

In the above mathematical equation there are two variables - one called A and the other called B. Any particular value can be given to B and the resulting equation determines the value of A.

In JOVIAL the variables are called items. Instead of using single letters for the name of a variable or item as was done in the mathematical equation, the item names in JOVIAL consist of two to six alphanumeric (letters and numbers) symbols with the first character always a letter.

Thus a JOVIAL statement similar to the mathematical equation shown previously might appear as follows:

```
ALPHA = BETA + 6 $
```

The two items here are called ALPHA and BETA. If BETA is given any particular value it determines the value of ALPHA.

The dollar sign ($) is used in JOVIAL as a punctuation character and it indicates the end of a JOVIAL statement. A period (.) is reserved for other uses in the JOVIAL language and thus the dollar sign was decided upon to indicate the termination of a statement.

To use an item in a JOVIAL program it must have been previously defined. This is frequently done at the beginning of the program. We shall discuss shortly what is meant by defining an item.

An alternate method of defining items in JOVIAL is to use what is called a "compool." A compool is essentially a list of item definitions to which the compiler has access when it translates the JOVIAL statements into binary.

There are many different kinds of items in JOVIAL and we will discuss each kind in due course. For the present, however, let us discuss what is called an integer type of item.

An integer type item is one which can hold whole numbers only - not mixed numbers or fractions.

Below is an example of an item definition for an integer type item:

ITEM ALPHA I 3 U $

To define an integer type of item the programmer would write the code word ITEM on his program coding paper. This would be followed by the name that the programmer wished to give the item: for example, ALPHA. Next he would write an I to indicate that he was defining an integer type item.

Next he would specify the number of binary positions he wanted to have allocated for the item. At this point in our study, a knowledge of the binary number system would be helpful. Let us only mention enough about the binary number system, however, to enable us to continue.

In a decimal number each digit position is ten times the value of the position to the right of it. For example, in a 3-digit decimal whole number, the rightmost position is the units (or 1's) position. The 2nd position from the right is the tens position. The 3rd position from the right is the hundreds position.

Thus a whole number in the decimal number system of 432 is really a shorthand representation of,

$$4 \times 100 + 3 \times 10 + 2 \times 1$$

or

$$400 + 30 + 2$$

A similar situation exists in the binary number system with the difference being that only two symbols are available, 0 and 1, and each symbol position is twice the value of the one to the right of it. For example, in a 3 bit* whole number, the rightmost position is the units (or 1's) position. The 2nd position from the right is the two's position. The 3rd position from the right is the four's position.

Thus a whole number in the binary number system of 101 is really a shorthand representation of,

$$1 \times 4 + 0 \times 2 + 1 \times 1$$

or

$$4 + 0 + 1$$

or equivalent to the number 5 in the decimal number system.

*The term bit is a contraction of the expression "binary digit."

Therefore if we wished to represent the number of week days in a week using the decimal number system, we would say that there are 5. If we desired to represent the same number of days using the binary number system, we would say that there are 101.

If there is confusion concerning which number system is being employed when a number is written, a subscript may be attached. Thus 101_2 is clearly a binary number (base 2, which means two symbols: namely, 0 and 1) and not a decimal number (base 10, which means ten symbols: namely, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) of one hundred and one.

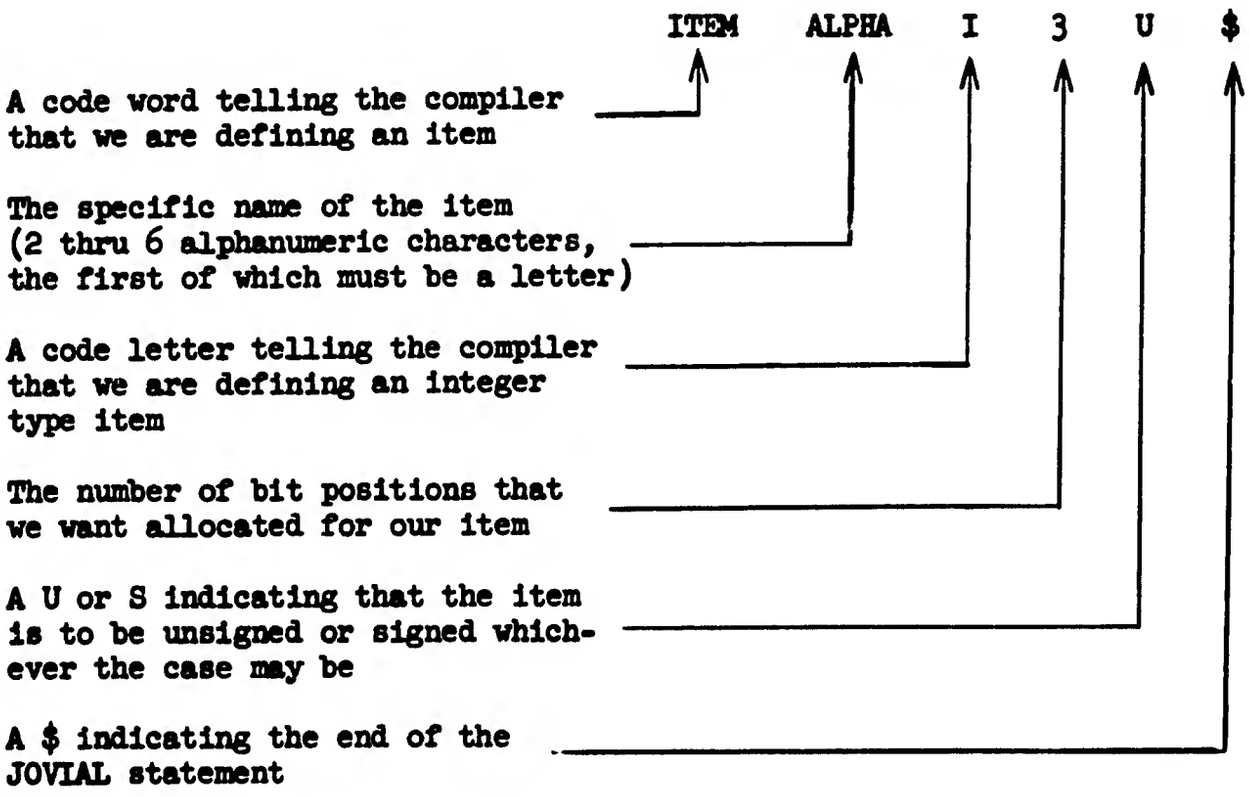
Returning to the portion of the item definition which specifies the number of bits, we might use 3 as the number of bits we want to allocate to our item. This would mean that item ALPHA would occupy a space of 3 binary positions. Thus ALPHA would be capable of holding numbers in binary from 000 through 111, or in other words the following binary numbers:

000	No four, no two, no one to give a decimal 0
001	No four, no two, a one to give a decimal 1
010	No four, a two, no one to give a decimal 2
011	No four, a two, a one to give a decimal 3
100	A four, no two, no one to give a decimal 4
101	A four, no two, a one to give a decimal 5
110	A four, a two, no one to give a decimal 6
111	A four, a two, a one to give a decimal 7

The last thing specified in the item definition of an integer type item is whether or not the contents of the item is to contain an algebraic sign. If the item would ever contain negative values, the programmer would specify S for "signed." If the item will always contain positive values and never negative values, the programmer would specify U for "unsigned."

The programmer then indicates the end of the JOVIAL statement by a dollar sign.

Thus in summary we have the following as the definition of item ALPHA:


ITEM ALPHA I 3 U $
A code word telling the compiler that we are defining an item
The specific name of the item (2 thru 6 alphanumeric characters, the first of which must be a letter)
A code letter telling the compiler that we are defining an integer type item
The number of bit positions that we want allocated for our item
A U or S indicating that the item is to be unsigned or signed whichever the case may be
A $ indicating the end of the JOVIAL statement

EXERCISES

2.1 a. Convert the following binary numbers to numbers in the decimal number system:

011
1011
1100
10101
1101

b. Given the item definition ITEM BETA I 4 U $

(1) What is the name of the item?

(2) What type of item is it?

(3) How many bits are being allocated for the item?

(4) What is the largest binary number that the item can contain?

(5) What decimal number does the answer to (4) above represent?

(6) What is the purpose of the dollar sign in the item definition?

c. How many bits should be allocated for an unsigned integer type item which can contain binary numbers up to the equivalent of the decimal number 49?

d. Define an unsigned integer type item and call it GAMMA. Make it large enough to hold binary numbers equivalent to the decimal number 127.

ANSWERS TO PRECEDING EXERCISES

2.1

a.

011 = 2+1 or 3
1011 = 8 + 2 + 1 or 11
1100 = 8 + 4 or 12
10101 = 16 + 4 + 1 or 21
1101 = 8 + 4 + 1 or 13

b.

(1) BETA

(2) An I type or integer type

(3) 4 bits

(4) 1111

(5) 1111 = 8 + 4 + 2 + 1 or 15

(6) It signifies the end of the JOVIAL statement

c. Since 49 = 32 + 16 + 1, the binary equivalent is 110001.

It is readily seen that this binary number requires 6 binary positions to represent it, hence the item should have 6 bits allocated for it.

d. ITEM GAMMA I 7 U $

We have seen how an item is defined in JOVIAL. The item is merely the empty vehicle. We need now to place some data into the item.

Let us define a signed integer type item called RHO. It is to be capable of containing positive and negative numbers the absolute value of which can be as large as 7.

The absolute value or absolute magnitude of a number is simply the number without the sign. Thus the absolute value of -4 is 4, the absolute value of +6 is 6, the absolute value of -7 is 7.

Since RHO will need to be capable of handling three binary positions for the maximum magnitude ($7_{10} = 111_2$) and will require one bit position for the sign, it should be defined as being 4 bits long.

```
ITEM RHO I 4 S $
```

Next, suppose we wish to place a value of 5 into RHO. This can be done with the following statement. It is called an assignment statement or set statement.

```
RHO = 5 $
```

The "=" should be interpreted as "set equal to." Thus the statement reads,

RHO set equal to 5.

Notice that in the assignment statement there is a left term (in this case RHO) and a right term (in this case 5).

The assignment statement says "compute the value of the right term and place that value into the left term."

The right term may be almost anything. For example, it can be a constant - as it was in the assignment statement above. Or it can be an item, or an arithmetic expression, or a few other things we will study later.

The left term must be an item. Later on we will modify this slightly to include a piece of an item or any variable but for now let us say that it must be an item. At any rate it must be a variable because if something is to be set into it, it must be a variable. It cannot be a constant.

In the JOVIAL statement RHO = 5 $ it is possible that item RHO already contained a value when the assignment statement was performed. This previous value in item RHO would now be destroyed and RHO would contain 5.

If we wished to set RHO to -3 we would write the following JOVIAL statement:

```
RHO = -3 $
```

EXERCISES

2.2 a. Given the following two item definitions:

```
ITEM NUMB I 5 S $
ITEM COST I 5 S $
ITEM TEMP I 5 S $
```

(1) Write a JOVIAL statement that will set item NUMB to the value 9.

(2) Write a JOVIAL statement that will set item COST to the value 4.

(3) Write a JOVIAL statement that will set item NUMB to the contents of item COST (Assume that you do not know what value is in either item).

(4) Write a JOVIAL statement that will set item COST to the contents of item NUMB (Assume that you do not know what value is in either item).

(5) What is the contents of items NUMB and COST after the following JOVIAL program has operated?

```
NUMB = 2 $
COST = 7 $
NUMB = COST $
```

(6) What is the contents of items NUMB and COST after the following JOVIAL program has operated?

```
NUMB = 5 $
COST = 11 $
TEMP = NUMB $
NUMB = COST $
COST = TEMP $
```

(7) Define two signed integer type items called ROSE and LILY. Make each capable of handling absolute values as large as 59. Then write a JOVIAL program that will exchange the contents of the two items. (You will also need to define an item to use for temporary storage).

(8) What is the absolute value of -37, of +63?

(9) Write a JOVIAL statement to set item NUMB to -8.

(10) Write a JOVIAL statement to set item COST to the negative of the largest absolute value that it can contain.

ANSWERS TO PRECEDING EXERCISES

2.2 a.

(1) NUMB = 9 $

(2) COST = 4 $

(3) NUMB = COST $

(4) COST = NUMB $

(5) NUMB contains the value 7

COST contains the value 7

Note that the statement NUMB = COST $ destroyed the previous contents of item NUMB (which was 2) and replaced it with the value contained in item COST (which was 7). Notice also that the contents of item COST is still intact.

(6) NUMB contains the value 11

COST contains the value 5

(7)
```
ITEM ROSE I 7 S $
ITEM LILY I 7 S $
ITEM TEMP I 7 S $
TEMP = ROSE $ )     (TEMP = LILY $
ROSE = LILY $ ) OR (LILY = ROSE $
LILY = TEMP $ )     (ROSE = TEMP $
```

(8) 37, 63

(9) NUMB = -8 $

(10) COST was defined as 5 bits. Since it is a signed item 1 bit is required for the sign. This leaves 4 bits for the magnitude of the number. Four binary positions can hold 1111_2 or 8 + 4 + 2 + 1 = 15. Therefore the JOVIAL statement would be

COST = - 15 $

The arithmetic operations in JOVIAL consist of addition, subtraction, multiplication, and division.

If it is desired to set an item ALPHA to 5 more than the contents of item BETA, the statement used would be

```
ALPHA = BETA + 5 $
```

or

```
ALPHA = 5 + BETA $
```

As was stated earlier, the right term of an assignment statement is computed and the result is placed into the left term.

If it is desired to set an item ALPHA to 7 less than the contents of item BETA, the statement used would be

```
ALPHA = BETA - 7 $
```

If we wished to set an item PI to three times the contents of item GAMMA, the statement used would be

```
PI = 3 * GAMMA $
```

or

```
PI = GAMMA * 3 $
```

Thus the symbol for multiplication that is used in JOVIAL is the asterisk (*). No implied multiplication is permitted. Thus it is <u>incorrect</u> to write

```
PI = 3 GAMMA $
```

If we wished to set an item ROSE to half of the contents of item BLUE, the statement used would be

```
ROSE = BLUE / 2 $
```

If BLUE is an odd number, the resulting computation of the right term will yield a number with a fraction. For example, suppose that BLUE contains the value 19. Then BLUE/2 will result in the value $9\frac{1}{2}$. However, in our discussion thus far, we have defined only integer type items. So assuming that ROSE has been defined as an integer type item, the only thing that would get into ROSE would be 9.

In summary then, the symbols for addition, subtraction, multiplication, and division are respectively +, -, *, and /.

EXERCISES

2.3 a. Assuming that two integer type items ALPHA and BETA have been previously defined,

(1) What is the contents of item BETA after the following program has operated?

```
ALPHA = 3 $
BETA = 4 $
ALPHA = BETA + 5 $
BETA = 3 * ALPHA $
```

(2) Write a program which will set item ALPHA to 5, and then set item BETA to 7 more than ALPHA.

(3) Write a JOVIAL statement that will set item ALPHA to 5 times the contents of BETA.

(4) Write a JOVIAL statement that will set item BETA to one-fifth of the contents of ALPHA.

b. What is wrong with each of the following JOVIAL statements?

(1) `ALPHA + 2 = BETA + 1 $`

(2) `BETA = 2 * ALPHA`

(3) `EPSILON = RHO + 9 $`

(4) `2ABC = ALPHA - 1999 $`

(5) `ROSE = 3 LILY $`

ANSWERS TO PRECEDING EXERCISES

2.3 a. (1) BETA contains 27

(2)
```
ALPHA = 5 $
BETA = ALPHA + 7 $
```

or

```
ALPHA = 5 $
BETA  = 7 + ALPHA $
```

or

```
ALPHA = 5 $
BETA = 12 $
```

(3)
```
ALPHA = 5 * BETA $
```

or

```
ALPHA = BETA * 5 $
```

(4)
```
BETA = ALPHA/5 $
```

b. (1) The left term of an assignment statement must be as item (variable) and not a constant or arithmetic expression.

(2) The dollar sign ($) signifying the end of the JOVIAL statement is missing.

(3) An item name must be from 2 to 6 alphanumeric characters and the item name EPSILON contains 7.

(4) An item name must start with a letter and the item name 2ABC starts with a number.

(5) Implied multiplication is illegal. There must be an asterisk (*) between the 3 and the item name LILY.

Sometimes it is desirable to increase the value in an item by a constant. The following statement will achieve this and is quite frequently used in JOVIAL programming.

```
ALPHA = ALPHA + 1 $
```

The right term is computed and the result is placed into the item specified by the left term. Thus the contents of ALPHA has been increased by 1.

The right term of an assignment statement may be a quite complex arithmetic expression. It may also contain parentheses if needed. Below are some various examples of assignment statements:

```
a.   ALPHA = 2*(BETA + 6)/GAMMA $

b.   ROSE = LILY*(BLUE/TUNIA + 9) + 15 $

c.   ANSWER = BPLACE*BPLACE - 7*APLACE*CPLACE $
```

The order of computing the result of an arithmetic expression is the same as it is in mathematics.

First the sub-result contained within a parenthesis is computed. Multiplications and divisions are performed first starting at the left and proceeding to the right. Then additions and subtractions are done.

In example a. above, if BETA contained the value 2 and GAMMA contained the value 4, the quantity in parenthesis would become 8. Then multiplying and dividing from left to right would give first 16 and then 4. Therefore the contents of ALPHA would become 4.

In example b., if LILY contained 3, BLUE contained 4, and TUNIA contained 2, the quantity in parenthesis would become 11. Then multiplying by LILY and adding 15 would result in 33+15 or 48. Thus ROSE would be set to 48.

In example c., if APLACE, BPLACE and CPLACE contain respectively 3, 4, and 5, item ANSWER would be set to 4*4-7*3*5 or 16 - 105 or - 89.

EXERCISES

2.4 a. Considering the items PRICE, COST, and OHEAD to be previously defined, what is the contents of PRICE after the following program has operated?

```
COST = 27 $
OHEAD = COST/9 $
PRICE = COST + 2 * OHEAD + 3 $
```

b. Considering the items to be previously defined, write a program statement that will set RESULT to 6 more than twice NUMBA.

c. Considering the items to be previously defined, write a program statement that will set RESULT to three times the sum of NUMBA and NUMBB.

d. Considering the items to be previously defined, write a program statement that will set AREA to 17 less than 5 times the product of LENGTH and WIDTH.

ANSWERS TO PRECEDING EXERCISES

2.4

a. PRICE contains 36.

b.
```
RESULT = 2 * NUMBA + 6 $
```

or

```
RESULT = 6 + 2 * NUMBA $
```

c.
```
RESULT = 3 * (NUMBA + NUMBB) $
```

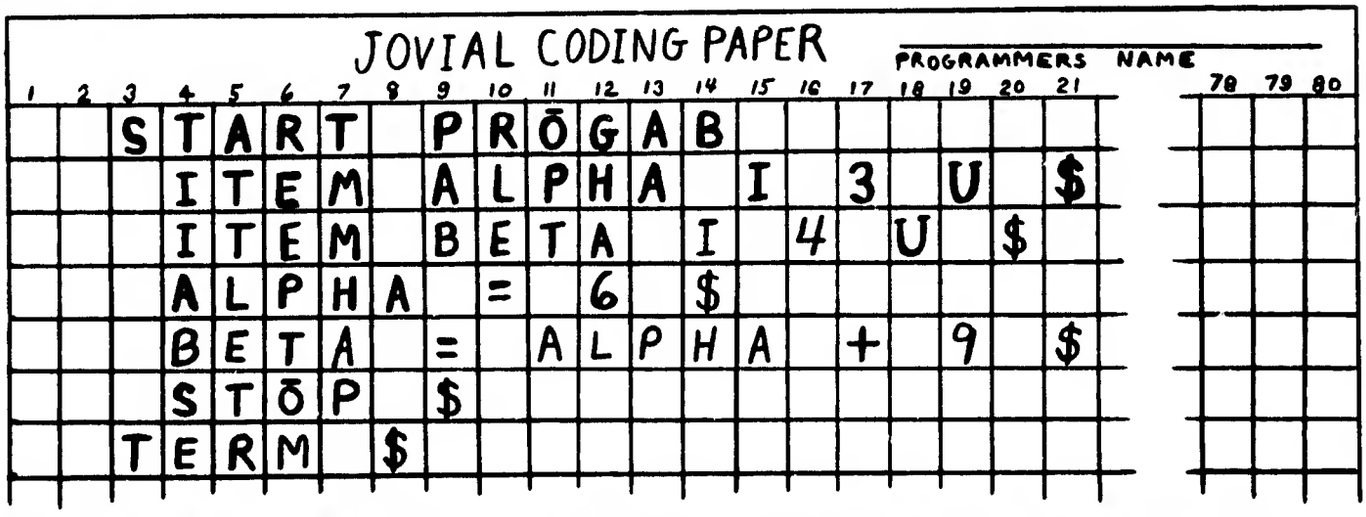
Obvious variations of the above are also correct.

d.
```
AREA = 5 * LENGTH * WIDTH - 17 $
```

CHAPTER 3

JOVIAL FORMAT ON CODING PAPER AND REPRESENTATION OF HOLLERITH INFORMATION ON PUNCHED CARDS

A program written on JOVIAL coding paper is shown below. The coding sheet is simply an array of little squares. There are 80 columns and about 25 lines. The 80 columns on the paper correspond to the 80 columns on an IBM punch card.



Coding may begin in any column on the sheet. Coding may stop in any column but must not extend beyond column 66. Columns beyond 66 are used to number cards and also frequently contain the program identity or name.

A JOVIAL statement may continue for as many lines as necessary. The end of any one line, however, cannot be the middle of a single entity, such as an item name. The break between lines must occur at a place where a space normally occurs.

Each line on the coding paper will be punched on a separate IBM card. Therefore, the program shown above would consist of 7 cards when punched. The cards are punched on typewriter-like machines called keypunches.

The first card in any JOVIAL deck must be a card beginning with the word START followed by the program name. There can be no JOVIAL statements on the START card. START is not a JOVIAL statement, therefore, although it may be followed by a dollar sign, no dollar sign is required.

Any remarks, preferably programmer's name, date, etc., should be added to the START card since this card is logged out on the printer by the compiler.

The last card in the deck must be TERM followed by a dollar sign.

Both the START card and the TERM card are instructions to the compiler program - the program that will translate the JOVIAL program to binary. They are not really part of the program but are necessary for compiling. The STOP instruction is one of the program statements and simply causes the computer to stop operating.

EXERCISES

3.1

a. How many columns are there on a sheet of JOVIAL coding paper? Why?

b. What must be the first card in a JOVIAL program?

c. What must be the last card in a JOVIAL program?

d. What two cards in a JOVIAL program are instructions to the compiler?

e. What is the last column on a JOVIAL coding sheet which may be used in coding a JOVIAL statement?

f. How many IBM cards would be required for punching up the following program?

JOVIAL CODING PAPER

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
    S T A R T   A D D S I X
    I T E M   N U M B R   I   1 Ø   S   $
    N U M B R   =   N U M B R   +   6   $
      S T Ø P   $
  T E R M   $
```

ANSWERS TO PRECEDING EXERCISES

3.1

a. The JOVIAL coding sheet has 80 columns to correspond to the 80 columns on an IBM card.

b. The first card in a JOVIAL program must be a card which contains the word START.

c. The last card in a JOVIAL program must be a card which contains the word TERM $.

d. The two cards in a JOVIAL program which are instructions to the compiler, are the START card and the TERM $ card.

e. The coding of a JOVIAL program statement cannot extend past column 66. Column 66 is a legitimate column to code in, but not 67, 68, etc.

f. Five cards would result from keypunching the program illustrated since each line is punched on a card.

Usually the JOVIAL coding sheets are given to a trained keypunch operator for card punching. Therefore, it is imperative that no ambiguity exist among the various symbols. For this reason, certain conventions have been established to prevent errors and confusion. This document will follow the three conventions listed below; namely,

1) On the coding sheet the number "zero" will be slashed, Ø, and the letter O will have a bar over it, $\overline{O}$.

2) On the coding sheet the number "one" will have a base attached to it, 1, and the letter "I" will be capped above and below, I.

3) The letter "Z" will have a bar in it, Z to distinguish it from the number "2."

The above three coding sheet conventions should eliminate major ambiguities if the programmer exercises care and neatness in writing his program on coding paper.

Some further rules associated with the JOVIAL programming language follow:

1) All numbers are written in the decimal number system unless otherwise indicated by the programmer. The manner in which the programmer indicates a non-decimal number (a number in the octal number system) will be described later.

2) Single terms, such as item names, constants, and operators, which consist of consecutive strings of letters or numbers, must not have any spaces between the first and last characters, and they cannot be broken into two consecutive cards.

   For example, if ALPHA is the name of an item, it cannot be coded as ALP HA with a space between the P and the H. Nor can AL be coded in columns 65 and 66 of one card with PHA in columns 1, 2, and 3 of the next card.

3) Where two consecutive entities which consist of consecutive strings of letters or numbers appear, they must be separated by at least one space. (e.g., ITEM ALPHA.....)

4) The end of a JOVIAL statement is signified by a dollar sign. It need not be preceded or followed by a space, but it may be preceded or followed by any number of spaces.

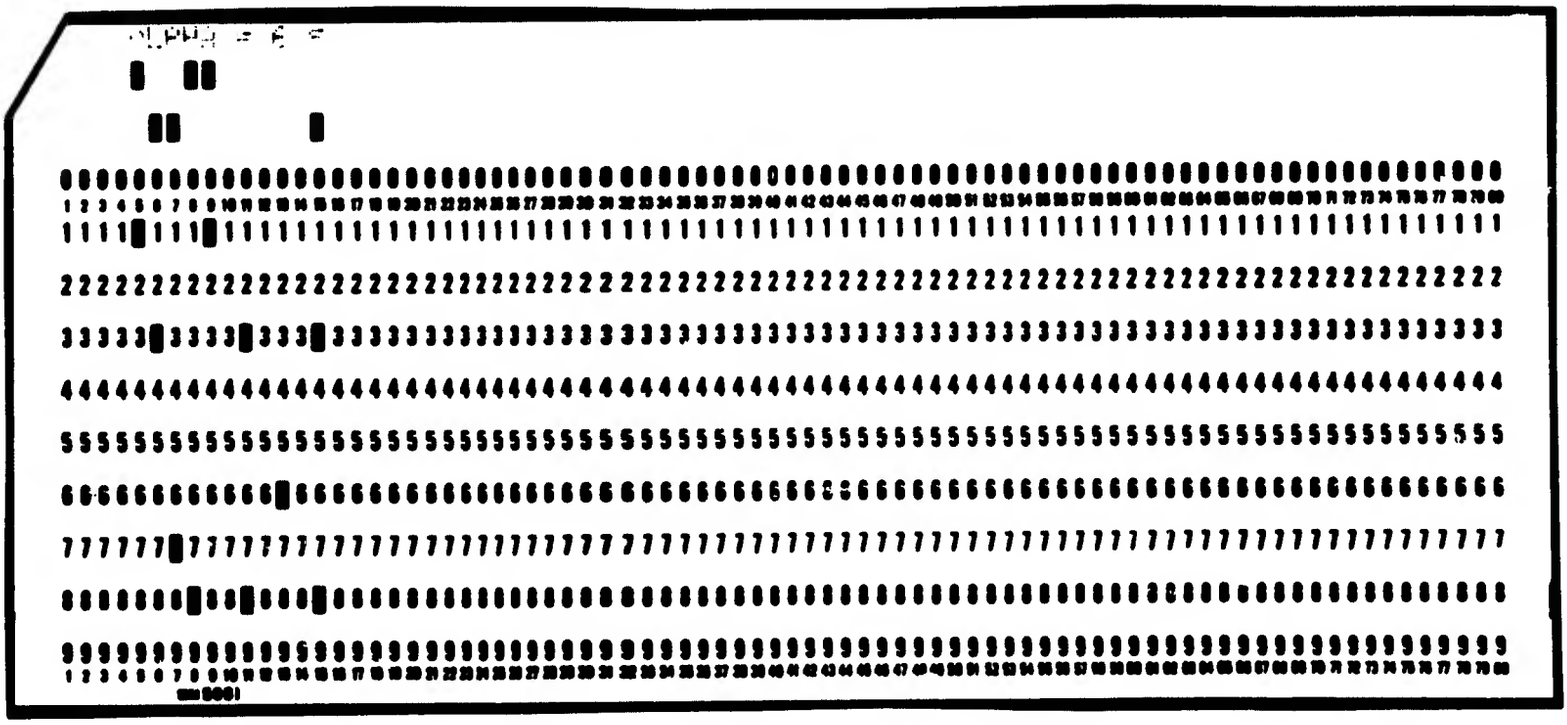
5) It is not necessary (although it might sometimes be desirable for the sake of readability) to separate special characters (non-alphanumeric) from single terms and (under some circumstances) to separate special characters from one another.

For example, in the following JOVIAL statement, no spaces need appear.

```
ALPHA=3*(BETA-GAMMA)/(EPSLON+RHO)$
```

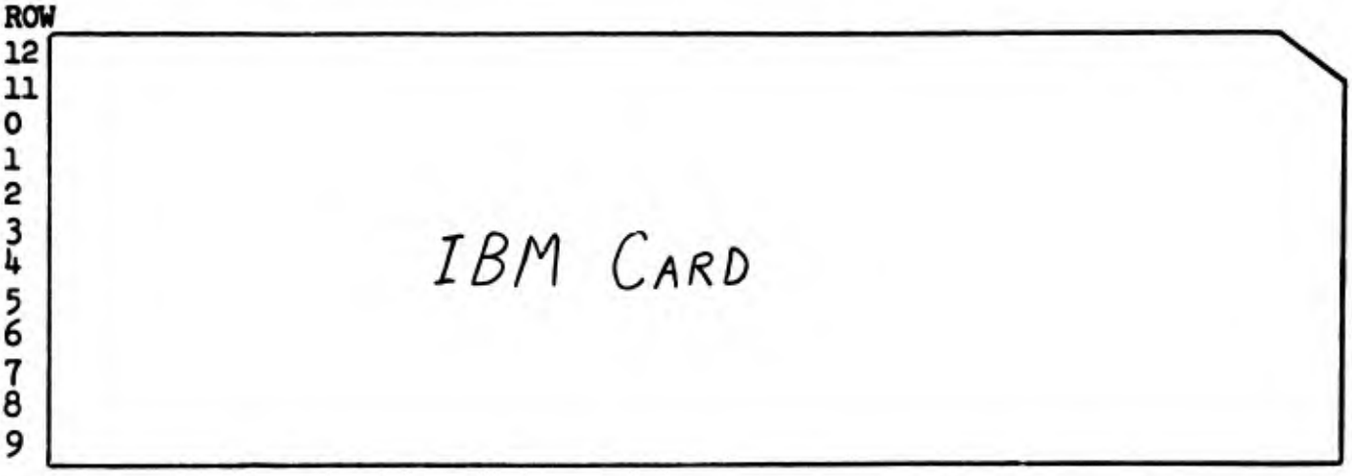
6) Wherever one space is required or permitted, any number of spaces may exist.

The JOVIAL statement ALPHA = 6 $ would appear on a punched card as shown below:



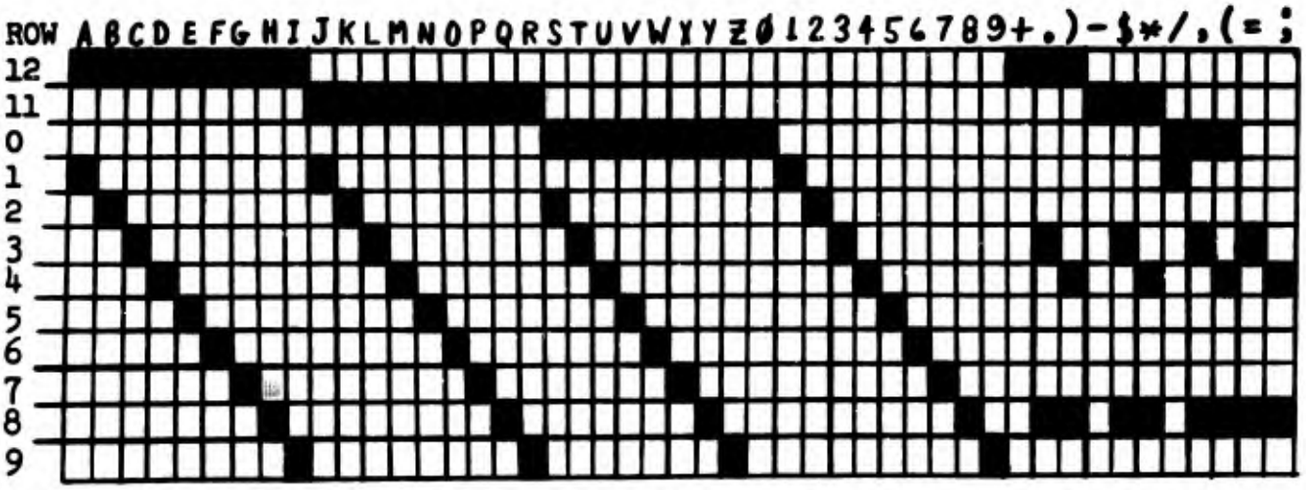
Observe that an A has been punched in column 5, an L in column 6, etc. The code which represents alphanumeric and puncutation - like data on cards is called "12-bit Hollerith Code."

A card column consists of 12 rows numbered from the top down as:



Note that the A punched in column 5 required punches in rows 12 and 1; the L required punches in rows 11 and 3, etc.

The following diagram gives the code for all of the Hollerith Characters:



From the diagram we see that punches in rows 11 and 8 would be required to represent the letter Q ir a particular column.

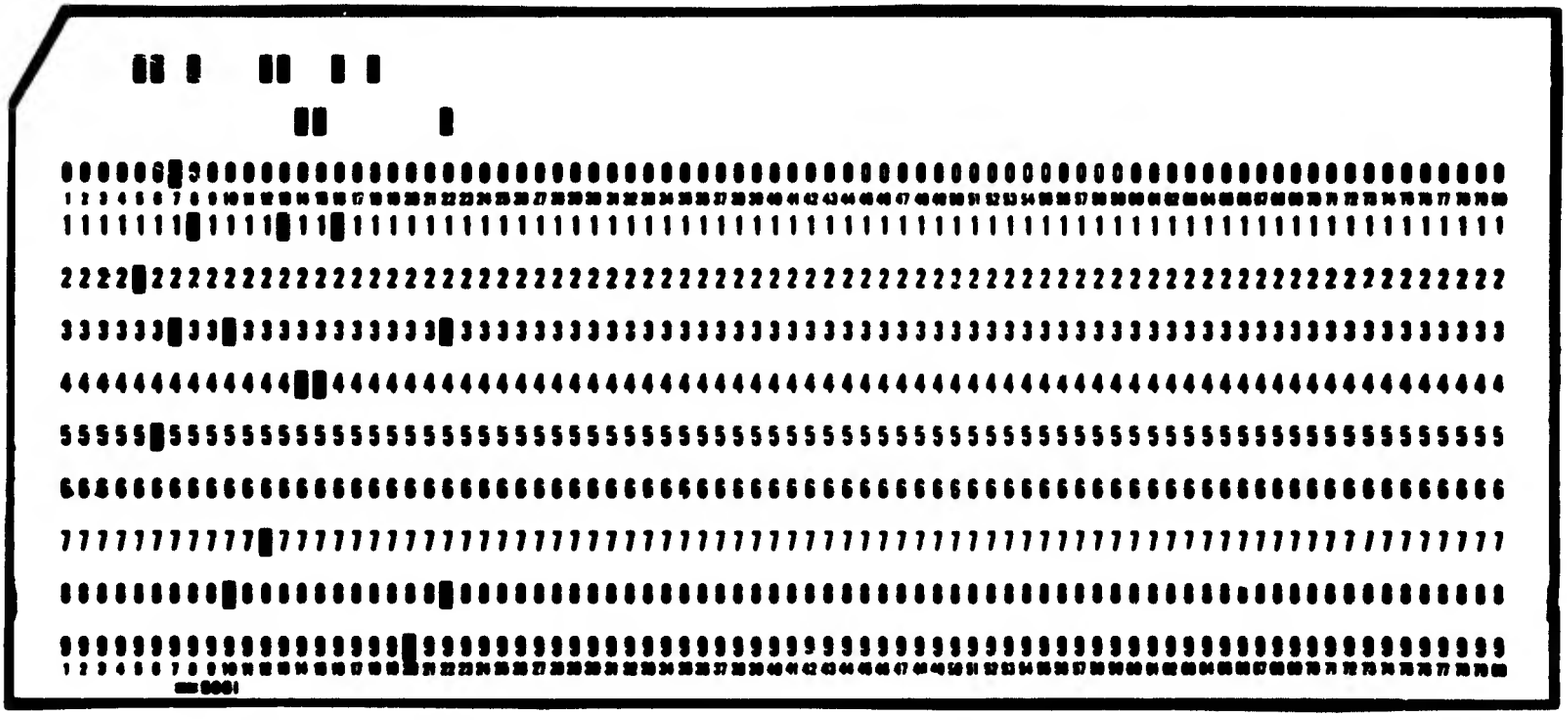
EXERCISES

3.2

a. What punches would be required in column 9 if we wished to represent a $ in that column?

b. What Hollerith Character would be represented by punches in rows 0 and 5 of a column?

c. What JOVIAL program statement is contained on the following card?



ANSWERS TO PRECEDING EXERCISES

3.2

a. In order to represent a $ in column 9, that column would require punches in rows 11, 3, and 8.

b. Punches in rows 0 and 5 of a particular column would represent the letter V in that column.

c. The JOVIAL statement represented on the card is BETA = GAMMA + 9 $

In summarizing this chapter, we may say that we have seen how a JOVIAL program is written on coding paper and subsequently how that same program is key-punched to be represented on punched cards.

The deck of cards containing the JOVIAL program is then submitted to the compiler. The compiler, you will recall, is a computer program that will translate the higher level language of JOVIAL into the low level language of 0's and 1's that the computer can understand and operate.

The compiler produces a punched deck containing the low level language. This deck is called a binary deck.

The binary deck is then submitted for operation. It is then that our program is operated. The first submission to the computer was merely for translation from the high level language to low level language.

Thus it essentially takes <u>two</u> submissions to the computer to get a JOVIAL program performed.

1. The first submission is for compiling.

2. The second submission is for running.

In the next chapter we will discuss JOVIAL decision-making statements. They are also known as IF statements.

CHAPTER 4

THE IF STATEMENT, THE GOTO STATEMENT, STATEMENT LABELS, AND COMPOUND STATEMENTS

The simple IF statement in JOVIAL is used when it is desirable to have a program make a decision between two possible courses of action.

For example, if we desire to add 3 to the contents of item ALPHA when ALPHA contains 5, or add 2 to ALPHA when ALPHA contains a value other than 5, we can use the following JOVIAL statements to do the job.

```
IF ALPHA EQ 5 $
   ALPHA = ALPHA + 1 $
   ALPHA = ALPHA + 2 $
```

The IF statement has the characteristic of being either true or false. In the above example the item ALPHA either contains 5 or it doesn't.

When the IF statement is true control proceeds with the next step in sequence, which in this case is ALPHA = ALPHA + 1 $. Then control continues on in the normal fashion with ALPHA = ALPHA + 2 $ being performed next, etc.

However, when the IF statement is false, control skips the statement immediately following the IF statement and proceeds with the second statement.

Thus we see in the example shown that when ALPHA contains 5, both succeeding statements will be performed, which results in adding 3 to the value of ALPHA. Also whenever ALPHA contains a value other than 5, only the second statement following the IF statement will be performed, ALPHA = ALPHA + 2 $, which results in only 2 being added to ALPHA.

EXERCISES

4.1 What does ALPHA contain after each of the following programs has operated?

a.
```
ALPHA = 6 $
IF ALPHA EQ 7 $
ALPHA = ALPHA + 1 $
ALPHA = ALPHA + 3 $
STOP $
```

b.
```
BETA = 29 $
ALPHA = BETA + 4 $
IF ALPHA EQ 33 $
ALPHA = ALPHA + BETA $
ALPHA = ALPHA + 1 $
STOP $
```

c.
```
BETA = 7 $
GAMMA = 9 $
ALPHA = GAMMA - BETA $
IF ALPHA EQ 2 $
ALPHA = BETA $
ALPHA = GAMMA $
STOP $
```

ANSWERS TO PRECEDING EXERCISES

4.1

a. ALPHA contains 9 since the IF statement is false and only the ALPHA = ALPHA+3 $ statement is performed.

b. ALPHA contains 63 since the IF statement is true and both of the statements following it are performed.

c. ALPHA contains 9. The IF statement is true, however, the second statement following the IF statement replaces the previous contents of ALPHA with the contents of GAMMA, thus in effect making the statement above it, ALPHA = BETA $, of no consequence.

The format for the simple IF statement is as follows:

IF [Left Term] relational operator [Right Term] $

The left term and right term may be almost anything. For example, they may be a constant, a variable, or an arithmetic expression.

Thus we may have

```
IF ALPHA EQ 2*BETA-6 $
```

where the left term is ALPHA and the right term is 2*BETA-6.

There are six different relational operators. We have seen one of them as EQ for "equal". The relational operators are

EQ - Equal
NQ - Not Equal
LS - Less Than
LQ - Less Than or Equal
GR - Greater Than
GQ - Greater Than or Equal

EXERCISES

4.2

a. If ALPHA contains 4 and BETA contains 6, is the following IF statement true or false?

```
IF ALPHA LS 2*BETA-1Ø $
```

b. If GAMMA contains 5 and BETA contains 19 is the following IF statement true or false?

```
IF BETA-GAMMA NQ 2*GAMMA+3 $
```

c. What does COUNT contain after the following program has operated?

```
COUNT = Ø $
NUMBR = 3 $
PARAM = 4 $
IF NUMBR EQ PARAM $
     COUNT = COUNT+1 $
     COUNT = COUNT+2 $
IF PARAM GR NUMBR $
     COUNT = COUNT+3 $
     COUNT = COUNT+4 $
     STOP $
```

d. Write a program which will examine item ABLE. If ABLE contains a value less than 9, add 7 to item ABLE, and if it contains 9 or greater, add 5 to item ABLE.

ANSWERS TO PRECEDING EXERCISES

4.2

a. The IF statement is false.

b. The IF statement is true.

c. COUNT contains 9 after the program operates.

d.
```
IF ABLE LS 9 $
   ABLE = ABLE+2 $
   ABLE = ABLE+5 $
   STOP $
```

It is frequently desirable to alter the sequence of operation in a series of JOVIAL statements. This can be done by use of the GOTO statement.

For example, suppose that if ALPHA contains 19 then we wish to add 7 to BETA and 9 to GAMMA. However, if ALPHA contains a value other than 19 then we wish to add 5 to EPSLON and 8 to RHO. The following program illustrates a solution.

```
        IF  ALPHA EQ 19 $
            GOTO LABELA $
            EPSLON = EPSLON+5 $
            RHO = RHO+8 $
EXIT.       STOP $
LABELA.     BETA = BETA+7 $
            GAMMA = GAMMA+9 $
            GOTO EXIT $
```

If the IF statement is true, control will go to the statement following it. Since this statement is a GOTO statement, control will then be transferred to the statement with the label LABELA. Thus 7 will be added to BETA, 9 will be added to GAMMA and control will go to the statement labeled EXIT.

Any operative instruction within the JOVIAL program may bear a statement label. Non-operative statements such as an item definition or the brackets START and TERM do not take statement labels.

A label consists of from 2 to 6 alphanumeric characters, the first of which must be a letter. The statement label must be followed by a period. No spaces may appear between the label and the period. Spaces may appear between the period and the statement itself, although no space is required.

A particular set of symbols may not appear as a statement label more than once in a main program. Statement labels may have the same set of symbols as those used for item names, although this practice is not recommended.

The GOTO statement may be used anywhere within a program to interrupt the normal sequence of control. The format consists of the expression, GOTO, the name of the statement label to which it is to go, and a dollar sign. GOTO is a single element of the statement and may not be written as two words.

Note also that only the statement label itself is written with a period; any reference to a statement label by a GOTO statement is written without the period.

When reference is made to a statement label by a GOTO statement, the program must contain a corresponding statement label. In other words, a GOTO statement must have some place to go.

EXERCISES

4.3 a. What is in COUNT after the following program operates:

```
       COUNT = Ø $
       BETA = 2 $
       ALPHA = BETA+3 $
       IF ALPHA LQ 5 $
          GOTO MORE $
          BETA = ALPHA $
          COUNT = COUNT+2 $
EXIT.  STOP $
MORE.  BETA = 2*ALPHA $
       COUNT = COUNT+ALPHA $
       GOTO EXIT $
```

b. What is in COUNT after the following program operates?

```
       COUNT = 2 $
       BETA = 7 $
AGAIN. IF BETA EQ 3 $
          STOP $
          BETA = BETA-1 $
          COUNT = COUNT * COUNT $
          GOTO AGAIN $
```

c. Write a program that will obtain a count in item COUNT of how many of the items ALPHA, BETA, and GAMMA contain quantities greater than 9.

d. What errors exist in the following program?

```
       ALPHA = Ø $
       BETA+1 = ALPHA+7 $
       IF BETA = 6
          GOTO JUNE. $
          ALPHA = ALPHA-2 $
          GOTO MORE $
     JUN BETA = BETA-7 $
     JUN ALPHA = BETA
          STOP $
```

ANSWERS TO PRECEDING EXERCISES

4.3 a. COUNT contains 5

b. COUNT contains 2^{16} or 65,536

c.
```
COUNT = ø $
IF ALPHA GR 9 $
   COUNT = COUNT+1 $
IF BETA GR 9 $
   COUNT = COUNT+1 $
IF GAMMA GR 9 $
   COUNT = COUNT+1 $
   STOP $
```

Note that setting COUNT to zero initially insures that COUNT will not contain miscellaneous information with which to begin. The clearing of items initially is referred to as "Initializing."

Note also that regardless of what ALPHA contains, we go on and examine BETA and GAMMA. Also, regardless of what ALPHA and BETA contain, we go on and examine GAMMA. Thus our COUNT could finally contain 0, 1, 2, or 3 depending upon the particular values in ALPHA, BETA, and GAMMA at the time of operation of the program.

d. (1) The second statement does not have a variable as the left term.

(2) There is no relational operator in the IF statement. It should be EQ and not "=."

(3) There is no dollar sign ending the IF statement.

(4) The first GOTO statement is incorrect because it has a period after the label describing where control should go. Furthermore, there is no label in the program by the name of JUNE.

(5) The second GOTO statement refers to a label called MORE and there isn't any statement in the program with that label.

(6) There are two statements with the same label (JUN). Also, neither one has a period after it as required for a statement label.

(7) The second statement which is labelled JUN is missing a dollar sign.

It was stated that the statement immediately following an IF statement is skipped when the IF statement is false.

Since it is frequently desirable to do several things when an IF statement is true, it would be advantageous to be able to treat several JOVIAL statements as one statement. This can be done by placing the brackets BEGIN and END around the several JOVIAL statements. The enclosed statements then become a compound statement.

Let us look at the first example in this chapter that we discussed, using this new concept. I will restate the problem.

The problem states that 3 is to be added to the contents of ALPHA when ALPHA contains 5, and 2 is to be added to ALPHA when ALPHA contains a value other than 5.

Using the idea of a compound statement, we can write our solution as follows:

```
    IF ALPHA EQ 5 $
       BEGIN
         ALPHA = ALPHA+3 $
         GOTO EXIT $
       END
         ALPHA = ALPHA+2 $
EXIT.  STOP $
```

Note that the brackets BEGIN and END do not take dollar signs. When the IF statement is true, control proceeds with ALPHA = ALPHA+3 $ and then continues with GOTO EXIT $ which takes us around the area which treats the false condition.

When the IF statement is false, control skips the first statement, which in this case is a compound statement, and proceeds with the statement ALPHA = ALPHA+2 $

A very common error that beginning JOVIAL programmers make, is that they do not program the bypassing of the false portion connected with an IF statement when that statement is true. They fall into the false portion from the true portion, thus, often performing both the true and false sections.

EXERCISES

4.4 a. What values are in ALPHA, BETA, and COUNT after the following program operates?

```
COUNT = Ø $
ALPHA = 1 $
BETA = 3 $
IF ALPHA+6 LS BETA+5 $
   BEGIN
     COUNT = COUNT+1 $
     ALPHA = BETA+2 $
     BETA = ALPHA+2 $
     GOTO OUT $
   END
     COUNT = COUNT+2 $
     ALPHA = 2*BETA+6 $
     BETA = 2*ALPHA-5 $
OUT. STOP $
```

b. Write a program which will set ROSE to 5 and MARY to 7 if BETTY contains 12. Otherwise set ROSE to 29 and MARY to BETTY.

ANSWERS TO PRECEDING EXERCISES

4.4 a.

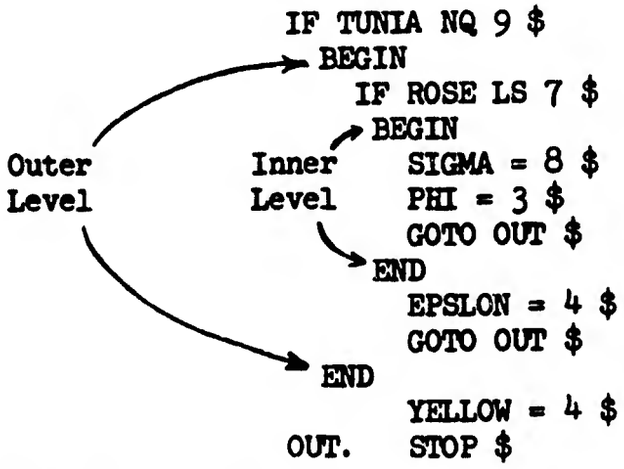
```
ALPHA contains 5
BETA contains 7
COUNT contains 1
```

b.

```
IF BETTY EQ 12 $
  BEGIN
     ROSE = 5 $
     MARY = 7 $
     GOTO OUT $
  END
     ROSE = 29 $
     MARY = BETTY $
OUT. STOP $
```

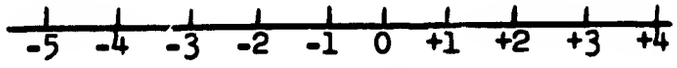
A compound statement may be labeled just as a simple JOVIAL statement may be labeled. The label may precede or follow the BEGIN. The label will actually be assigned by the compiler to the first simple JOVIAL statement within the compound statement. This in effect labels the compound statement.

Also, a compound statement may contain a compound statement. Thus, we have BEGIN and END brackets within BEGIN and END brackets. Another way of stating this is to say that compound statements may be nested. Thus, we can speak of different levels of compound statements.



The thing to remember is that there must be an END for each BEGIN.

When working with relational operators for comparing quantities, it is well to keep relationships as found on a number line in mind.



Number Line

On the number line, the number 0 separates the positive numbers from the negative numbers. Also, numbers get larger as one moves to the right. Thus, -3 is larger than -4 and +3 is larger than +2.

This gives us an idea of how we can check for negative values if we so desire. The following IF statement will differentiate between positive and negative numbers.

```
IF NUMBR LS Ø $
```

Assuming the quantity to be examined is in item NUMBR, the IF statement is true for negative numbers and false for positive numbers.

EXERCISES

4.5

a. Given an item called MCAND and one called MLIER, each of which contains a non-negative whole number (either or both can be zero) and assuming that we cannot use the * or multiplication feature of JOVIAL, write a program which will obtain the product of MCAND and MLIER and place it into item PRODCT. Note: Multiplication is a process of repeated additions.

b. Test your solution to the above problem using the following table.

	MCAND	MLIER	EXPECTED RESULT PRODCT	ACTUAL RESULT PRODCT
Trial 1	0	2	0	
Trial 2	3	0	0	
Trial 3	3	2	6	
Trial 4	1	4	4	

ANSWERS TO PRECEDING EXERCISES

4.5 a.

```
      PRODCT = Ø $
      IF MCAND EQ Ø $
        GOTO EXIT $
AGAIN. IF MLIER EQ Ø $
 EXIT.   STOP $
        PRODCT = PRODCT+MCAND $
        MLIER = MLIER-1 $
        GOTO AGAIN $
```

b. The column containing "Actual Result" should contain the same values as the column called "Expected Result".

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

System Development Corporation,
Santa Monica, California

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

SELF-INSTRUCTIONAL JOVIAL MANUAL - CHAPTERS 1, 2, 3 & 4.

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(Last name, first name, initial)*

Cutler, D. I.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
15 October 1965	47	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
U. S. Government Contracts	SP-2214
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES

This document has been cleared for open publication and may be disseminated by the Clearing House for Federal Scientific & Technical Information.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

Presents a manual for JOVIAL programming language training which does not require any previous programming knowledge on the part of the student. This arrangement presents the advantages of sensitivity to individual learning abilities, and independence from a formal class. Each chapter contains review exercises followed by the answers to the questions presented in the exercises. Chapter I is an introduction to JOVIAL, Chapter II describes items, the assignment statement and the four basic arithmetic operations of JOVIAL. Chapter III presents JOVIAL coding formats and a representation of Hollerith information on punched cards. Chapter IV discusses the "If" and the "Goto" statements, statement labels and compound statements. (author)

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
JOVIAL						
Programming						
Language						
Arithmetic						
Coding						
Hollerith						
Training						
Manual						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP.** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.