C582-PROG-ST

AIR TRAINING COMMAND

STUDENT TEXT

JOVIAL SELF INSTRUCTIONAL MANUAL

OZR 0123
OZR 0123-1
OZR 0123-3

May 1966

RESPONSIBLE AGENCY OF ATC
Keesler Technical Training Center
Keesler AFB, Mississippi

DESIGNED FOR ATC COURSE USE ONLY

ABOUT STUDENT STUDY GUIDES AND STUDENT WORKBOOKS

Student study guides and student workbooks are designed by the Air Training Command as student training publications for use in training situations peculiar to courses of this command. Each is prepared for a particular Unit of Instruction as reflected in the course syllabus.

The **STUDENT STUDY GUIDE** *contains the specific information required in the Unit of Instruction or it will refer to other publications which the student is required to read. It contains the necessary information which is not suitable for student study in other available sources. The material included or referred to is normally studied either outside the classroom or during supervised study periods in the classroom. Also included are thought-provoking questions which permit self-evaluation by the student and which will stimulate classroom discussion.*

The **STUDENT WORKBOOK** *contains the specialized job procedures, important information about the job, questions to be answered, problems to be solved and/or work to be accomplished by the student during the classroom/laboratory, airplane/missile/equipment activity. It serves as a job sheet, operations sheet, mission card, check list or exercise to be performed during classroom or laboratory periods. Also included are questions which will aid the student in summarizing the main points of the Unit of Instruction.*

The **STUDENT STUDY GUIDE AND WORKBOOK** *is a training publication which contains both student study guide and student workbook material under one cover.*

Since this publication is DESIGNED FOR ATC COURSE USE ONLY and must not conflict with the information and/or procedures currently contained in Technical Orders or other official directives, it is updated frequently to keep abreast of changes in qualitative training requirements. Students who are authorized to retain this publication after graduation are cautioned not to use this material in preference to Technical Orders or other authoritative documents.

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) Independance from a formal class.

Computer Systems Department
KTTC, Mississippi

JOVIAL SELF INSTRUCTIONAL MANUAL

CONTENTS

Chapter	Title
1	Introduction to JOVIAL
2	Items, The Assignment Statement, and the Four Basic Arithmetic Operations
3	JOVIAL Format on Coding Paper and Representation of Hollerith Information on Punched Cards
4	The IF Statement, the GOTO Statement, Statement Labels and Compound Statements
5	Tables, Subscripted Items, and Flow Diagrams
6	The FOR Statement
7	Status Items, Hollerith Items, The Exchange Statement and the Test Statement

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.

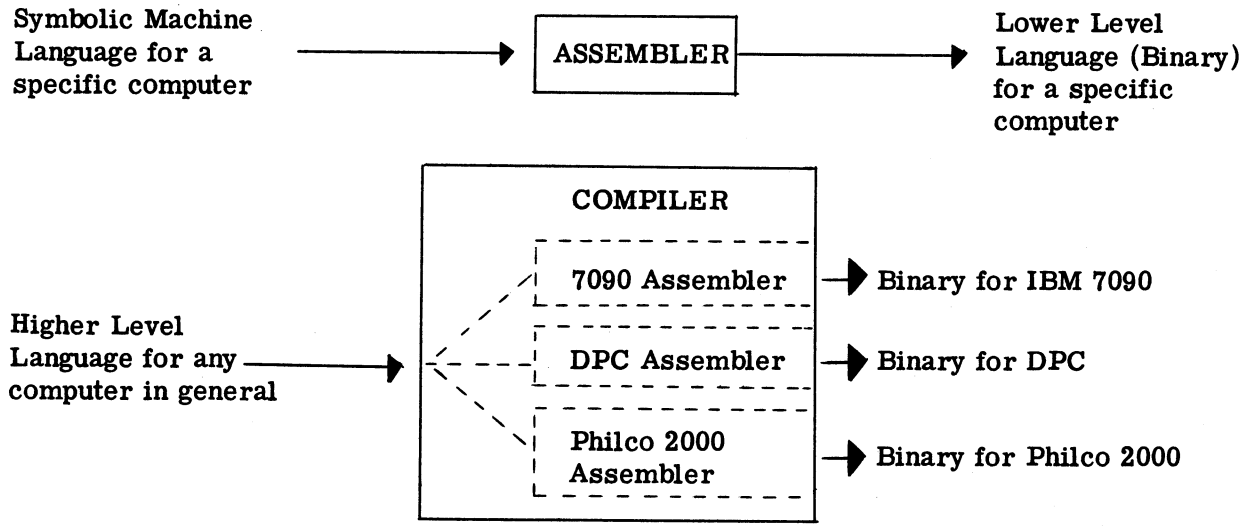
JOVIAL is a high level language. Other higher level languages are ALGOL, MAD, FORTRAN, COBOL, NELIAC, etc. JOVIAL was developed by the System Development Corporation to be used for programming large scale command and control systems. Work on the language began in 1959. Since that time, many modifications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

To which level does the JOVIAL programming language belong?

_____

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?

_____

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

_____

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

The three levels of programming language are:

low level language

intermediate level language

high level language

JOVIAL is a high level programming language.

Assembler

Compiler

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 or more 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 A 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 "A" to indicate that he was defining an arithmetic 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,

$$4 \times 100 + 3 \times 10 + 2 \times 1 \quad \text{or,}$$

$$400 + 30 + 2$$

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,

$$1 \times 4 + 0 \times 2 + 1 \times 1 \quad \text{or}$$

$$4 + 0 + 1$$

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

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.

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

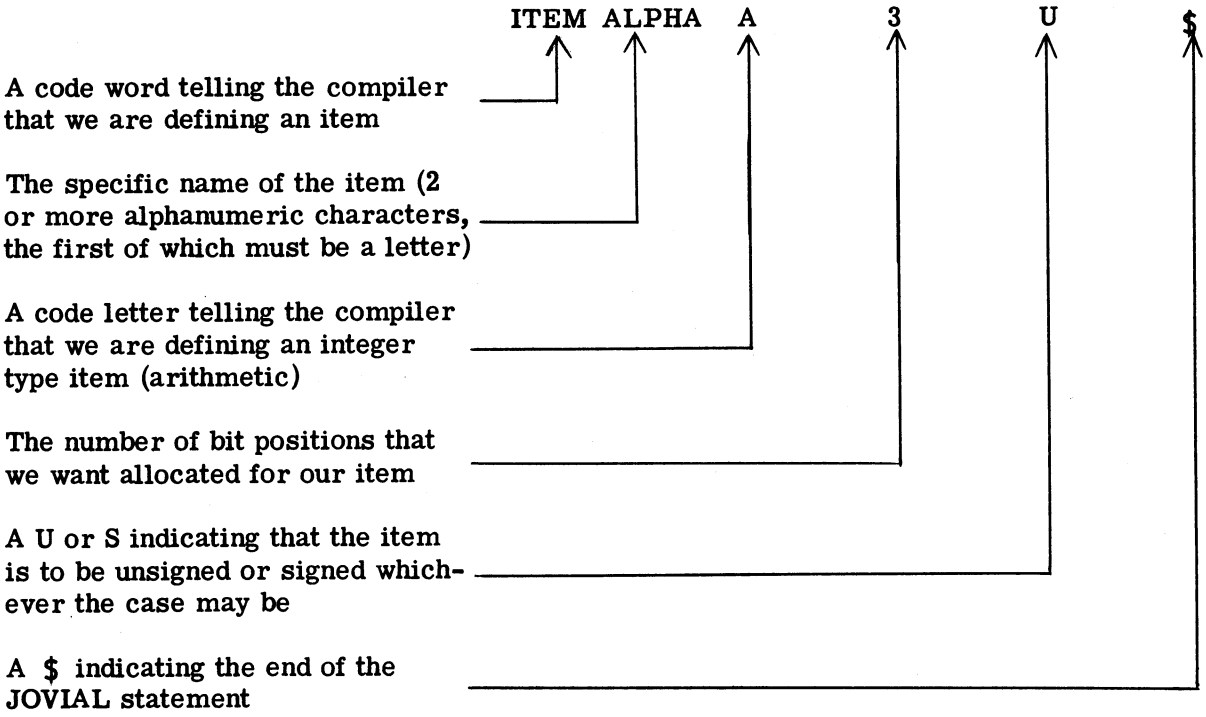
Returning to the portion of the item definition which specifies the number of bits, we might use 3 as the number of bits we want to allocate to our item. This would mean that item ALPHA would occupy a space of 3 binary positions. Thus ALPHA would be capable of holding numbers in binary from 000 through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



EXERCISES

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

011
1011
1100
10101
1101

Given the item definition ITEM BETA A 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?

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?

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

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

(1) BETA

(2) An A 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

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.

ITEM GAMMA A 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 A 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

Given the following three item definitions:

```
ITEM NUMB A 5 S $
ITEM COST A 5 S $
ITEM TEMP A 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? What is the absolute value 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 A 7 S $

ITEM LILY A 7 S $

ITEM TEMP A 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 incorrect 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 9-1/2. 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  $
```

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

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

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

at is wrong with each of the following JOVIAL statements?

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

```
BETA = 2  *  ALPHA
```

```
EPSILON = RHO + 9  $
```

```
2ABC = ALPHA - 1999  $
```

```
ROSE =3 LILY $
```

ANSWERS TO PRECEDING EXERCISES

BETA contains 27

```
ALPHA = 5  $
```

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

```
ALPHA = 5  $
```

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

```
ALPHA = 5  $
```

```
BETA = 12  $
```

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

```
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) The statement contains no error with the J-3 compiler.

(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

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 $
```

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

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

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

PRICE contains 36.

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

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

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

Obvious variations of the above are also correct.

```
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.

JOVIAL CODING PAPER

Programmer's Name

```
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35          78 79 80
                S  T  A  R  T
                I  T  E  M     A  L  P  H  A     A     3     U     '  '
                I  T  E  M     B  E  T  A     A     4     U     $     $
                A  L  P  H  A     =     6     $
                B  E  T  A     =     A  L  P  H  A     +     9     $
                S  T  O  P     $
             T  E  R  M     $
```

Coding may begin in any column on the sheet after and including column 17. Coding may stop in any column. Columns 1 through 16 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

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 first 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		17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
			S	T	A	R	T																
				I	T	E	M		N	U	M	B	R		A		1	Ø		S		$	
				N	U	M	B	R		=		N	U	M	B	R		+		6		$	
					S	T	Ō	P		$													
			T	E	R	M		$															

ANSWERS TO PRECEDING EXERCISES

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 can begin in column 17.

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 0 will have a bar over it, $\overline{0}$.

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, Ƶ 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 any number of spaces.

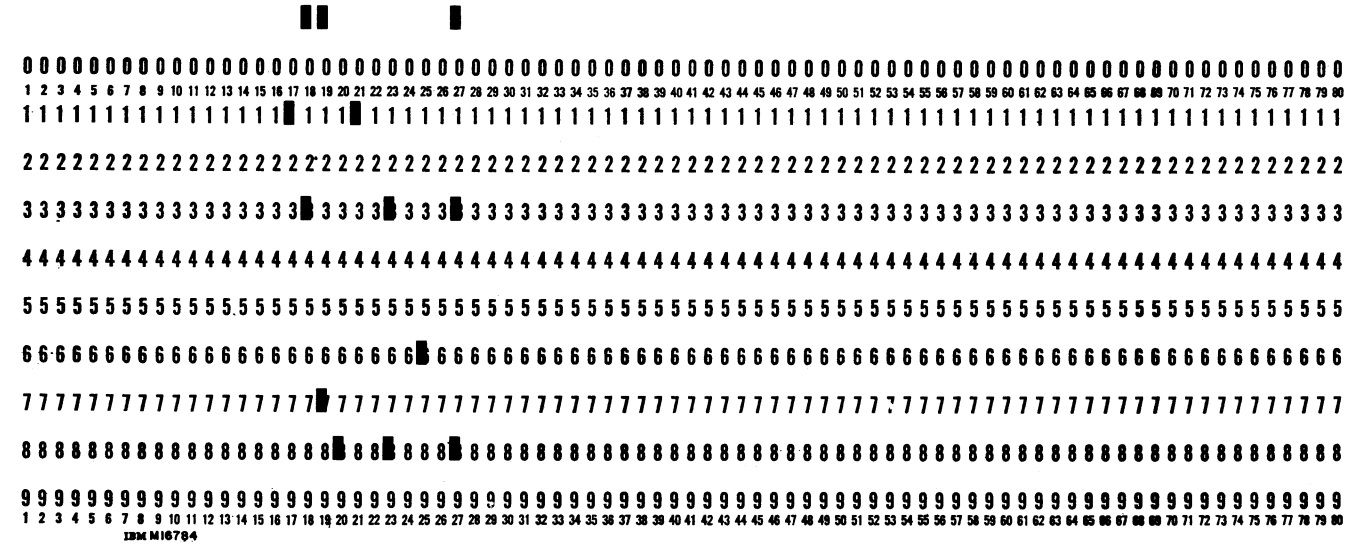
5) It is not necessary (although it might sometimes be desirable for the sake of readability) to separate special characters (non-alphanumeric) from single terms and (under some circumstances) to separate special characters from one another.

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

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

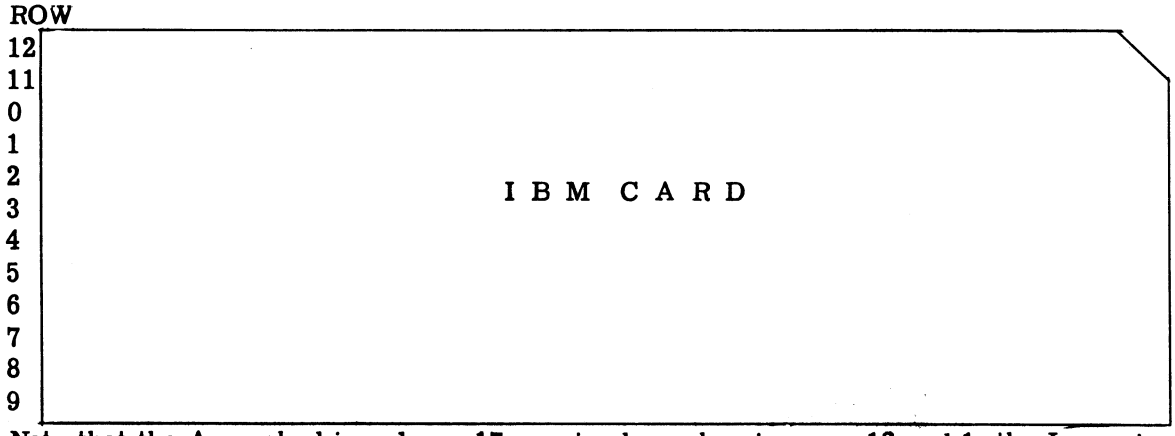
6) Wherever one space is required or permitted, any number of spaces may exist.

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



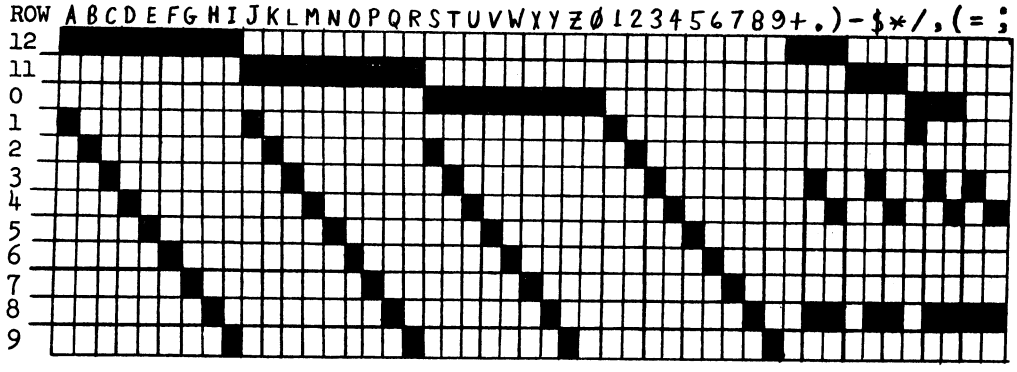
Observe that an A has been punched in column 17, an L in column 18, etc. The code which represents alphanumeric and punctuation - like data on cards is called "12-bit Hollerith Code."

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



Note that the A punched in column 17 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:



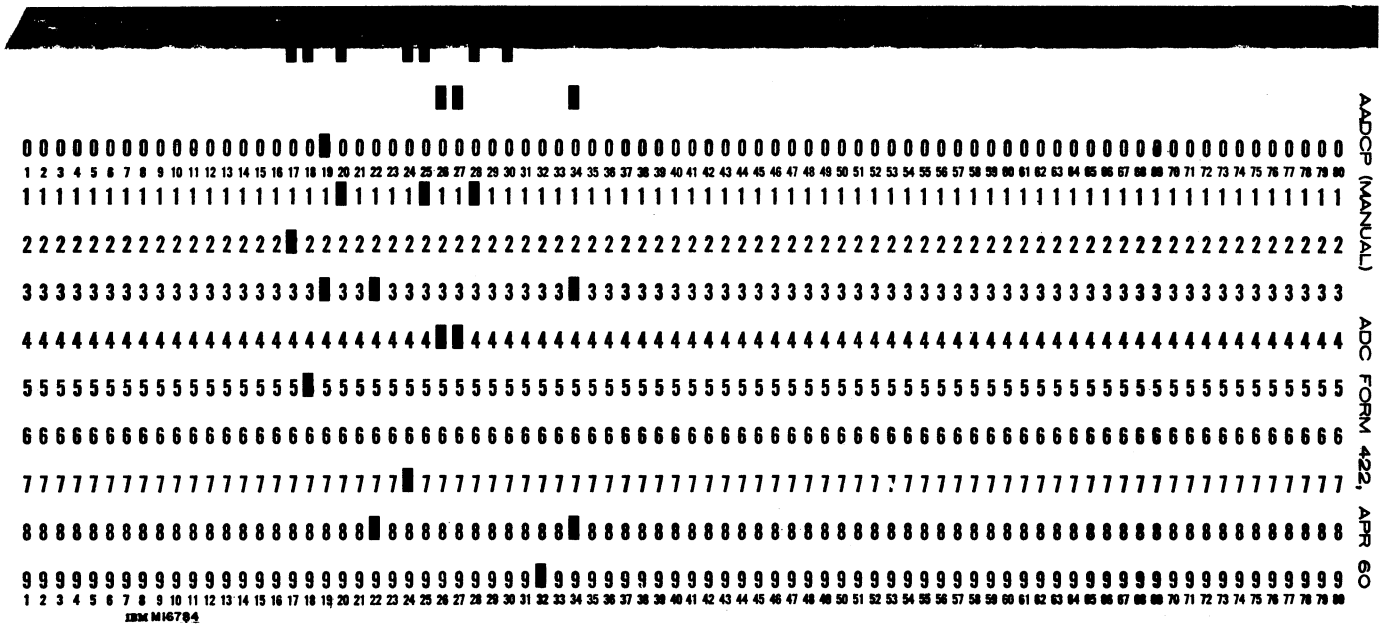
From the diagram we see that punches in rows 11 and 8 would be required to represent the letter Q in a particular column.

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.

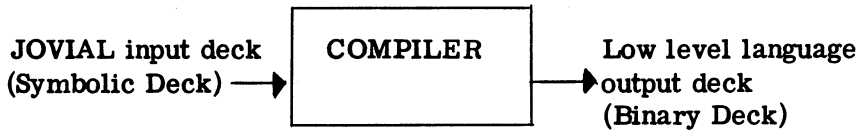
c. The JOVIAL statement represented on the card is BETA = GAMMA + 9 $

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 keypunched 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 two 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 + 7 $
   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 2 or more 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 $
IN.     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

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 should 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.

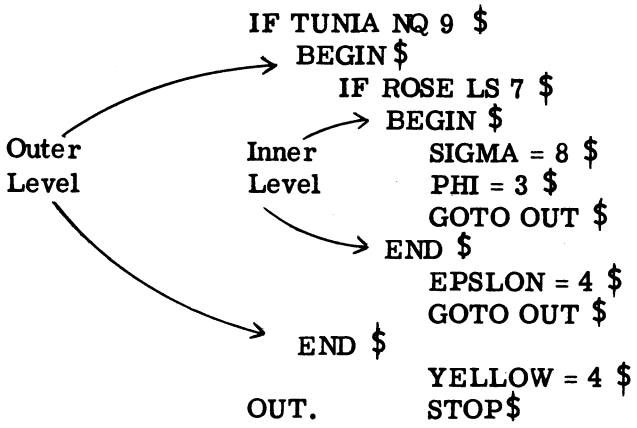
ANSWERS TO PRECEDING EXERCISES

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 $
```

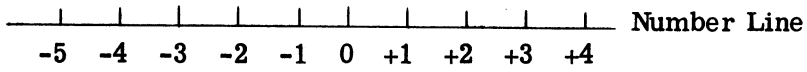
A compound statement may be labeled just as a simple JOVIAL statement may be labeled. The label may precede or follow the BEGIN. The label will actually be assigned by the compiler to the first simple JOVIAL statement within the compound statement. This in effect labels the compound statement.

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.



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.

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

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

ANSWERS TO PRECEDING EXERCISES

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

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

CHAPTER 5

TABLES, SUBSCRIPTED ITEMS, AND FLOW DIAGRAMS

We have mentioned that the variables manipulated in JOVIAL are contained in entities called items.

Frequently the data associated with a problem will not consist of a few different items, but rather, a large set of similar items. We can then arrange the items in what is called a table.

Suppose for example we wish to write a program which will add a set of ten numbers. We can design a table containing ten entries of the item. Let us arbitrarily call the table NUMØ and the item in each entry NUMBR.

We can think of our table appearing as shown below:

TABLE NUMØ

Item NUMBR	359	entry #Ø	First entry
Item NUMBR	4Ø85	entry #1	Second entry
Item NUMBR	26	entry #2	Third entry
Item NUMBR	732	entry #3	Fourth entry
Item NUMBR	18	entry #4	Fifth entry
Item NUMBR	437	entry #5	Sixth entry
Item NUMBR	2372	entry #6	Seventh entry
Item NUMBR	Ø	entry #7	Eighth entry
Item NUMBR	11	entry #8	Ninth entry
Item NUMBR	655	entry #9	Tenth entry

Observe that the first entry is designated entry #Ø, the second is entry #1, etc. This could present a confusion factor unless the student clearly fixes this idea in his mind.

In the example above, the value of 359 is contained in entry #Ø of item NUMBR, the value 4085 is contained in entry #1 of item NUMBR, etc.

This table NUMØ and its item NUMBR, can be defined in the compool (the dictionary-like table to which the compiler has access) or it can be defined at the beginning of our program. At any rate, it must be defined if we are to write a program which uses it.

Below is the table and item definition as it would be made in a program:

```
  T A B L E   N U M Ø   R   I Ø   $
    B E G I N
    I T E M   N U M B R   A   1 6   S   $
    E N D
```

The code word TABLE is written first. It tips off the compiler that a table definition is to follow. Then follows the unique name of the table, which in the above example is NUMØ. Table names consist of two or more alphanumeric characters, the first of which must be a letter.

The third field in a table definition contains either an R or a V. There are two types of tables in JOVIAL - those with a fixed (rigid) number of entries and those with a variable number of entries.

If we specify R (as in the example above) this means that the table will always contain the same number of entries.

If we specify V, this means that the number of entries in the table may vary from time to time as a result of programming the addition or deletion of entries.

For the present, we will discuss only tables consisting of a fixed number of entries (R-type tables). The V-type table will be taken up in full detail later.

Following the R or V in the table definition is the number of entries that the table is to have. In an R-type table this is the actual number of entries; whereas, in a V-type it is the maximum number ever expected.

The 1Ø specified in the example declares that the table NUMØ is to contain 1Ø entries.

Following the table declaration statement, the items to be defined for each entry are enclosed within a BEGIN-END bracket. Only those items enclosed within this BEGIN-END bracket are considered to be in the table defined.

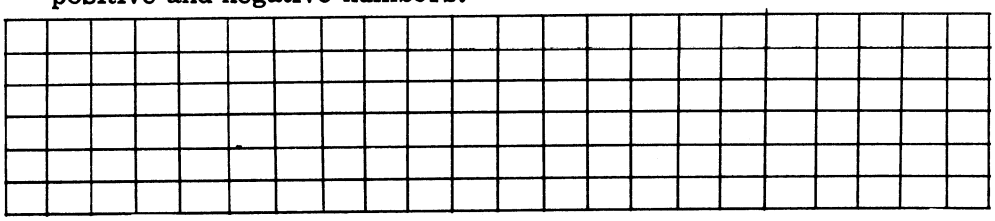
The item definition is made the same as described in Chapter 2. Each entry of a table must have at least one item. Many tables, however, have several items in each entry.

EXERCISES

5.1 a. In the table NUMØ of the preceding text (page 30) what is the content of entry #6 of item NUMBR?

b. In the table NUMØ of the preceding text, what is the content of the third entry of item NUMBR?

c. Define a table called TABØ which is to consist of 99 entries. The table is to have a fixed number of entries. Each entry is to contain the item ALPHA. Mak ALPHA an integer type item consisting of 16 bits and capable of containing positive and negative numbers.



ANSWERS TO PRECEDING EXERCISES

5.1 a. Entry #6 of item NUMBR contains 2372.

b. The third entry (entry #2) of item NUMBR contains 26.

c.

	T	A	B	L	E		T	A	B	Ø		R		9	9		$				
		B	E	G	I	N														
		I	T	E	M		A	L	P	H	A		A		1	6		S		$	
		E	N	D																	

Since all of the items NUMBR in the table NUMØ, described previously, are indistinguishable from one another, we must have some reference number when referring to item NUMBR in a particular entry.

We will attach the entry number to an item when speaking of the item in a particular entry. Thus, we might speak of entry #5 of item NUMBR as $NUMBR_5$.

The 5 is referred to as a subscript. However, specifying a subscript a little below the line of writing as done above does not lend itself to coding on coding paper and punching on punched card. In these instances everything is on the line.

Therefore, some symbology was developed to indicate a subscript. If we wish to refer to entry #5 of item NUMBR, or in other words, $NUMBR_5$, we will write it in JOVIAL as NUMBR ($ 5 $). Thus, the symbology of ($ $) merely indicated a subscript.

There can be no spaces in NUMBR ($5 $) since it refers to a single entity. The item with a subscript attached is called a subscripted variable or subscripted item.

If we wished to set entry #5 of item NUMBR to 678 we could do so with the following JOVIAL assignment statement:

```
NUMBR ( $ 5 $ ) = 678$
```

If we wished to add 9 to the present value in entry #8 of item NUMBR we could write:

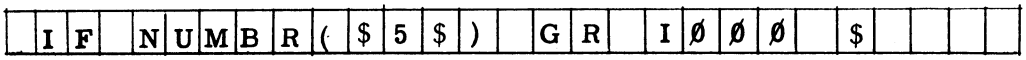
```
NUMBR ( $ 8 $ ) = NUMBR ( $ 8 $ ) + 9$
```

If we wished to determine whether or not entry #9 of item NUMBR contained a value larger than 1000 we could write:

```
IF NUMBR ( $ 9 $) GR 1ØØØ $
```

It is not permissable to leave off the subscript when specifying entry #Ø of an item.

when coding a JOVIAL statement on coding paper to write only 1 symbol in a box.



EXERCISES

The following questions pertain to table NUMØ described in the preceding text.

a. What value is contained in NUMBR ($ 4 $)?

b. What value is contained in NUMBR ($ Ø $)?

c. Write a JOVIAL statement that will set the value 39 into entry #3 of item NUMBR.

d. Write a JOVIAL statement that will add 789 to the contents of entry #6 of item NUMBR.

e. Write a JOVIAL statement that will replace the contents of entry #1 of NUMBR with the contents of entry #7 of NUMBR.

f. Write a JOVIAL statement that will place the sum of the first three entries of NUMBR into the single item TOTAL. By a "single item" I mean one that is not in a table. Consider TOTAL to be already defined.

g. Write a JOVIAL program that will clear each entry (set it to zero) of NUMBR if it contains a value larger than 5ØØ.

ANSWERS TO PRECEDING EXERCISES

a. Entry #4 of NUMBR contains 18.

b. NUMBR ($ 0 $) refers to entry #Ø, hence, 359.

c. NUMBR ($ 3 $) = 39 $

d. NUMBR ($6 $) = NUMBR ($ 6 $) + 789 $

e. NUMBR ($ 1 $) = NUMBR ($ 7 $) $

f. TOTAL = NUMBR ($ Ø $) + NUMBR ($1 $) + NUMBR ($ $) $

g.
```
IF NUMBR ( $Ø $ ) GR 5ØØ $
   NUMBR ( $Ø $ ) = Ø $
IF NUMBR ( $1 $ ) GR 5ØØ $
   NUMBR ( $1 $ ) = Ø $
IF NUMBR ( $2 $ ) GR 5ØØ $
   NUMBR ( $2 $ ) = Ø $
```

```
IF NUMBR ( $ 3$ )  GR 5ØØ $
   NUMBR( $3$ )  = Ø $
IF NUMBR ( $4$ )  GR 5ØØ $
   NUMBR ( $4$ )  = Ø $
IF NUMBR ( $5$ )  GR 5ØØ $
   NUMBR ( $5$ )  = Ø $
IF NUMBR ( $6$ )  GR 5ØØ $
   NUMBR ( $6$ )  = Ø $
IF NUMBR ( $7$ )  GR 5ØØ $
   NUMBR ( $7$ )  = Ø $
IF NUMBR ( $8$ )  GR 5ØØ $
   NUMBR ( $8$ )  = Ø $
IF NUMBR ( $9$ )  GR 5ØØ $
   NUMBR ( $9$ )  = Ø $
```

In problem 5.2g, it may be observed that the solution consisted of twenty JOVIAL statements and yet there were essentially only two basic statements involved. Each pair occurred ten times with only the subscript being different in each pair.

By using an item (variable) as a subscript, we can expand the power of the subscript tremendously.

Let us look at a program that will compute the sum of all entries of item NUMBR and place that sum in single item TOTAL.

```
          TABLE NUMØ R 1Ø $
            BEGIN
              ITEM NUMBR A 16 S $
            END
            ITEM TOTAL A 16 S $
            ITEM COUNT A 16 S $
            TOTAL = Ø $
            COUNT = Ø $
AGAIN.      TOTAL = TOTAL + NUMBR (  $COUNT $  ) $
            COUNT = COUNT + 1 $
            IF COUNT LS 1Ø $
            GOTO AGAIN  $
            STOP  $
```

In the above program the item NUMBR is subscripted by another item (variable) called COUNT. Notice that COUNT will take on all of the values from Ø through 9 before the program stops. Thus, TOTAL will contain the sum.

EXERCISES

5.3 The following problems involve table NUMØ described in the preceding text. Consider COUNT and TOTAL to be already defined.

a. What does TOTAL contain after the following program stops?

```
          COUNT = 1 $
          TOTAL = Ø $
AGAIN.    TOTAL = TOTAL + NUMBR ( $ COUNTS $ ) $
          COUNT = COUNT + 2 $
          IF COUNT LS 1Ø $
            GOTO AGAIN $
            STOP $
```

b. Using a variable subscript called COUNT, write a program that will clear each entry (set it to zero) of item NUMBR if it contains a value larger than 500.

c. Using a variable subscript called COUNT write a program that will examine each entry of item NUMBR and obtain a sum of the positive quantities in single item POSSUM and a sum of the negative quantities in single item NEGSUM. Consider POSSUM and NEGSUM to be already defined.

ANSWERS TO PRECEDING EXERCISES

).3 a. The program obtains a sum in item TOTAL of every other entry of item NUMBR starting with entry #1. In other words, the following entries of item NUMBR are summed; entries #1, 3, 5, 7 and 9. If this is applied to the specific illustration presented earlier, we get 4,085 + 732 + 437 + Ø + 655. Therefore, TOTAL would contain 5,909.

b.
```
          COUNT = Ø $
AGAIN .   IF NUMBR ( $ COUNT $ ) GR 5ØØ $
          NUMBR ( $ COUNT $ ) = Ø $
          COUNT = COUNT + 1 $
          IF COUNT LS 1Ø $
          GOTO AGAIN $
          STOP $
```

c.
```
          POSSUM = Ø $
          NEGSUM = Ø $
          COUNT = Ø $
AGAIN.    IF NUMBR ( $ COUNT $ ) GR Ø $
            BEGIN
              POSSUM = POSSUM + NUMBR ( $ COUNT $ )$
              GOTO UPCNT $
            END
              NEGSUM = NEGSUM + NUMBR ( $ COUNT $ ) $
UPCNT.        COUNT = COUNT + 1 $
              IF COUNT LS 1Ø $
              GOTO AGAIN $
              STOP $
```

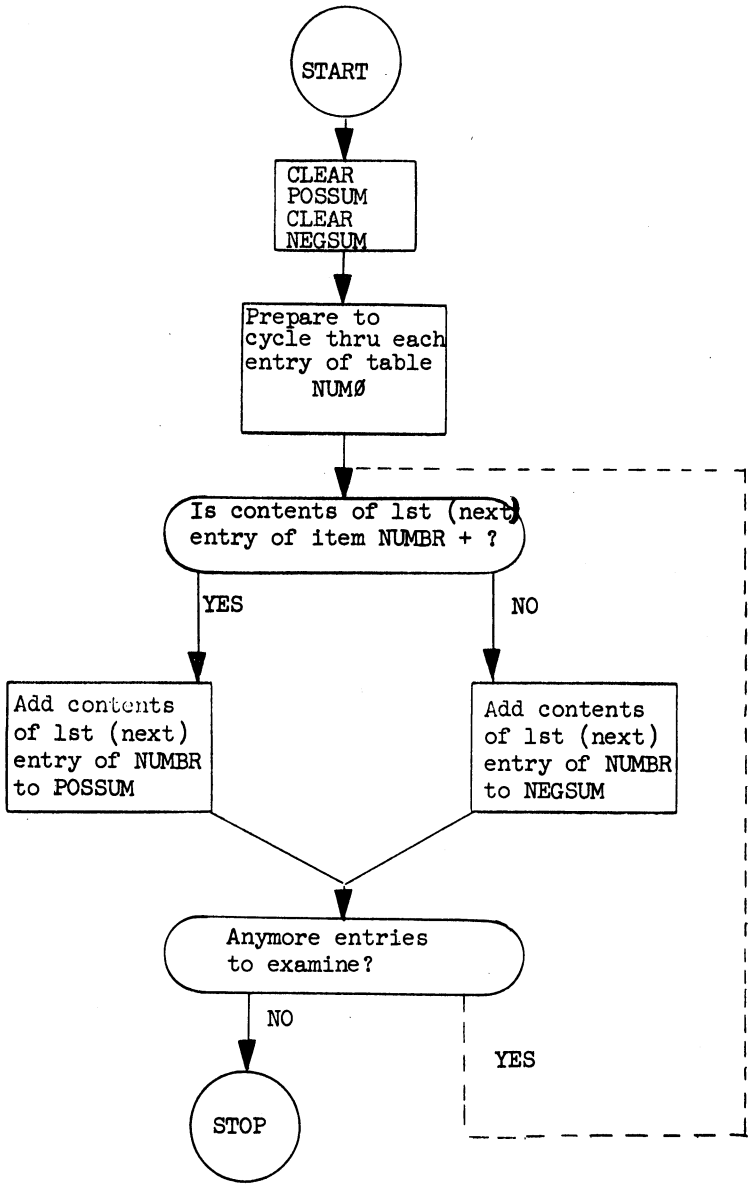
This appears to be a good time to bring up the subject of flow diagramming.

When the solution to a programming problem is very simple the programmer is able to contain the solution logic in his head. However, when the solution logic is more complex, it is a good idea to make a flow diagram before coding.

Let us flow diagram problem 5.3c in order to get the feel of what is involved in making a flow diagram.

For convenience I will again state the problem.

PROBLEM: Using a variable subscript called COUNT, write a program that will examine each entry of item NUMBR and obtain a sum of the positive quantities in single item POSSUM and a sum of the negative quantities in single item NEGSUM. Consider POSSUM, NEGSUM, COUNT, and item NUMBR in table NUMØ to be already defined.



Observe that decision boxes are "sausage shaped" and that non-decision statements are placed in boxes of a rectangular shape. A dash line signifies flow in the reverse direction.

Flow diagramming is somewhat of a personal thing and although there have been many efforts in the past to standardize flow diagramming symbols and techniques, the efforts have been largely unsuccessful.

A person will usually continue to flow diagram in the manner in which he first learned to flow. Then again, there are some programmers who never flow a solution - however, it is highly recommended that the art (or science) of flow diagramming be used whenever possible.

Before continuing, let me add that the flow can be made equally well going from left to right rather than from top to bottom. Also if the flow is lengthy you may want to employ "connectors" rather than forward lines or reverse lines (dash lines). Connectors are merely little circles that are labeled to tell where the flow goes.

Following is the same flow diagram shown above but with JOVIAL coding attached and a connector used to indicate the return flow (page 38).

EXERCISES

5.4 a. Given an item called MCAND and one called MLIER, each of which may contain a positive whole number or zero and assuming that we cannot use the JOVIAL multiplication operator "*", make a flow diagram of a program which will obtain the product of MCAND and MLIER and place it into item PRODCT. (Note: See problem 4.5a)

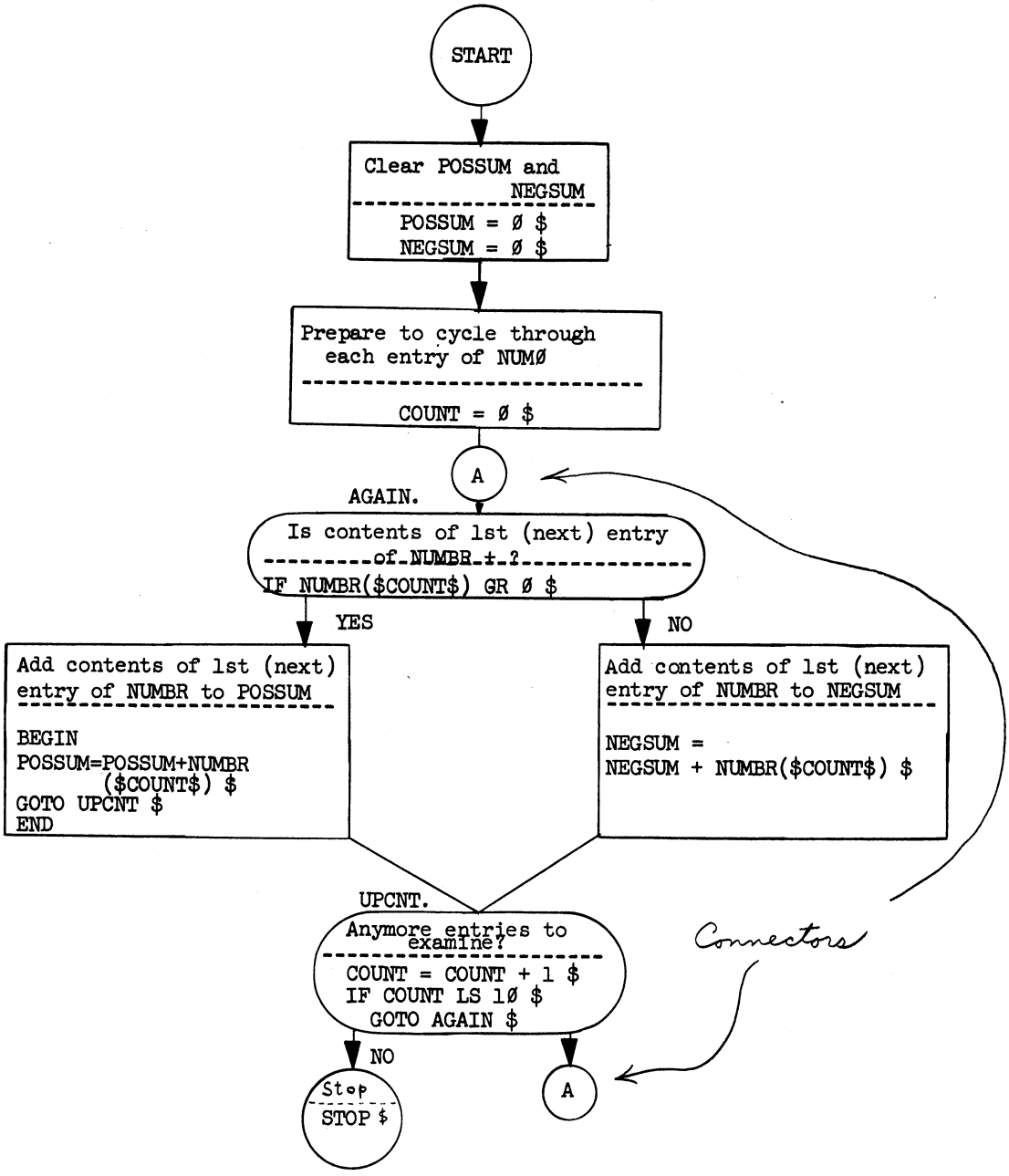
b. Given the following, make a flow diagram which can be used to determine how much money a man has in his pocket:

(1) He has 4 coins and their amounts are contained in item COIN of 4 entry table MONEY.

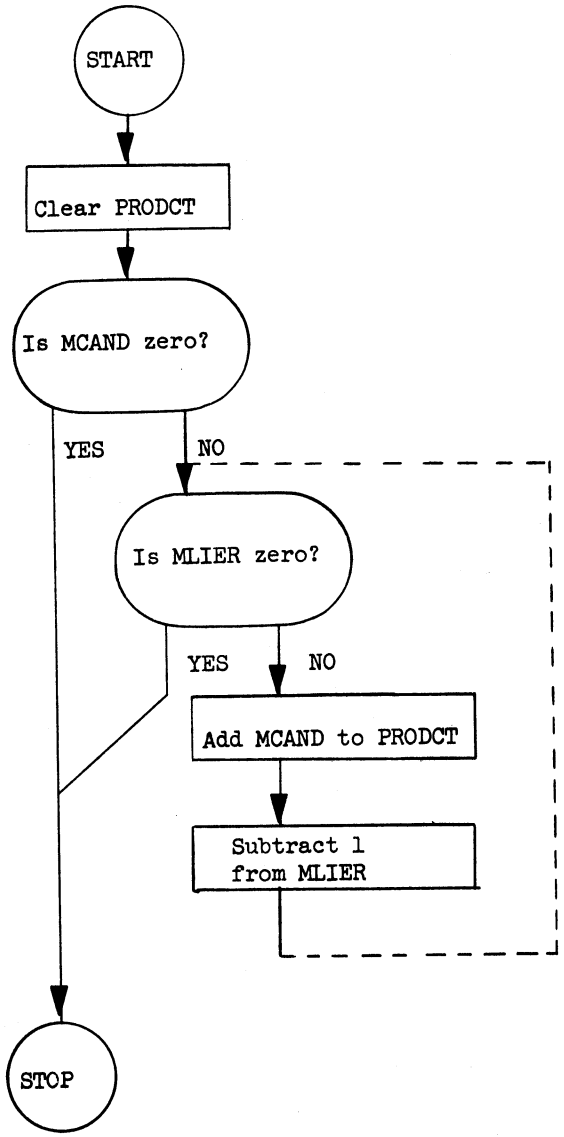
(2) The coins may possibly be duplicated.

(3) The coin possibilities are a penny, a nickel, a dime, a quarter, and a half-dollar.

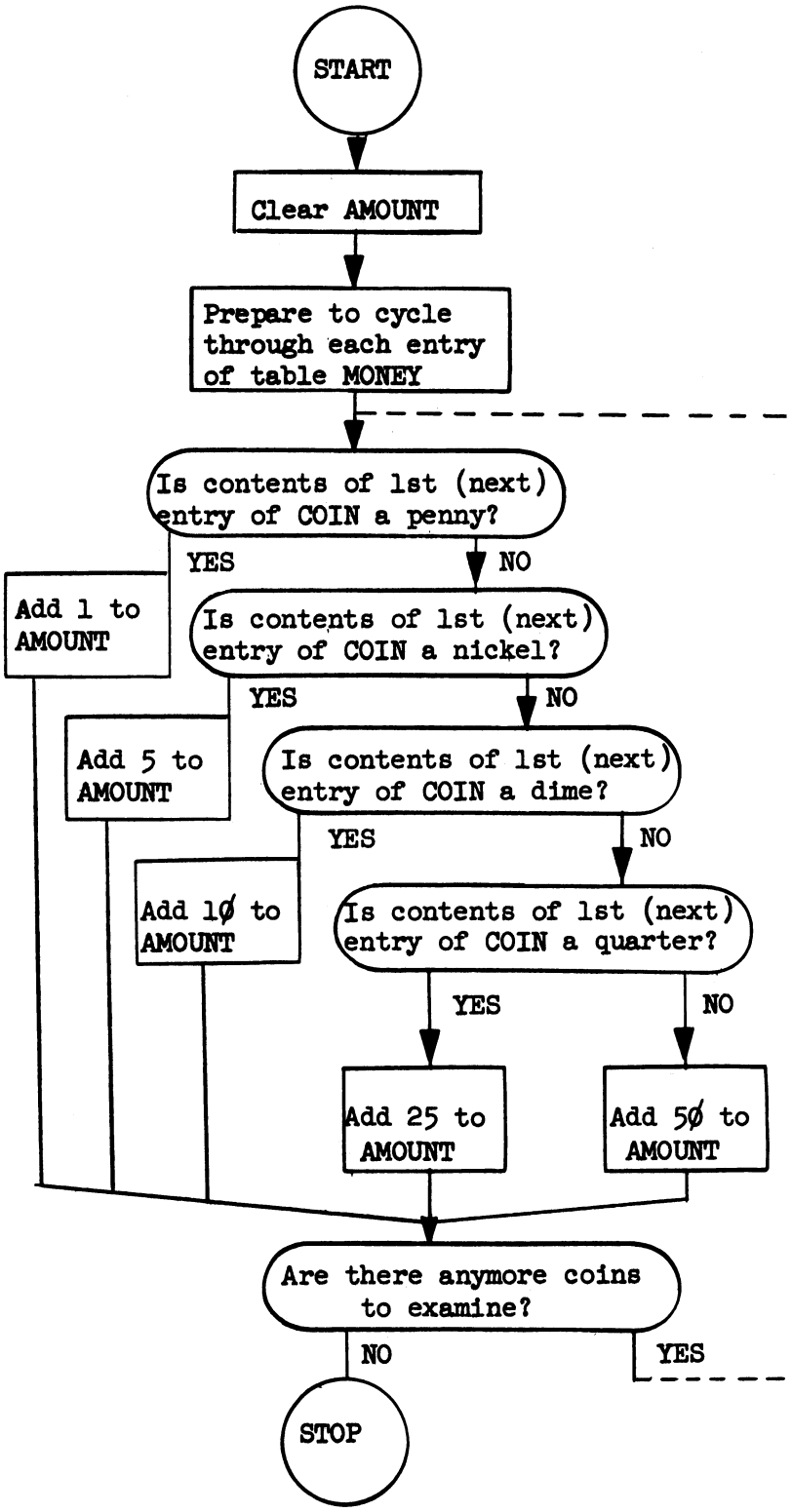
Set the total amount of his money into the single item AMOUNT.


START
Clear POSSUM and NEGSUM
POSSUM = Ø $
NEGSUM = Ø $
Prepare to cycle through each entry of NUMØ
COUNT = Ø $
A
AGAIN.
Is contents of 1st (next) entry of NUMBR + ?
IF NUMBR($COUNT$) GR Ø $
YES
NO
Add contents of 1st (next) entry of NUMBR to POSSUM
BEGIN
POSSUM=POSSUM+NUMBR ($COUNT$) $
GOTO UPCNT $
END
Add contents of 1st (next) entry of NUMBR to NEGSUM
NEGSUM = NEGSUM + NUMBR($COUNT$) $
UPCNT.
Anymore entries to examine?
COUNT = COUNT + 1 $
IF COUNT LS 1Ø $
GOTO AGAIN $
NO
Stop
STOP $
A
Connectors

ANSWER TO PRECEDING EXERCISE

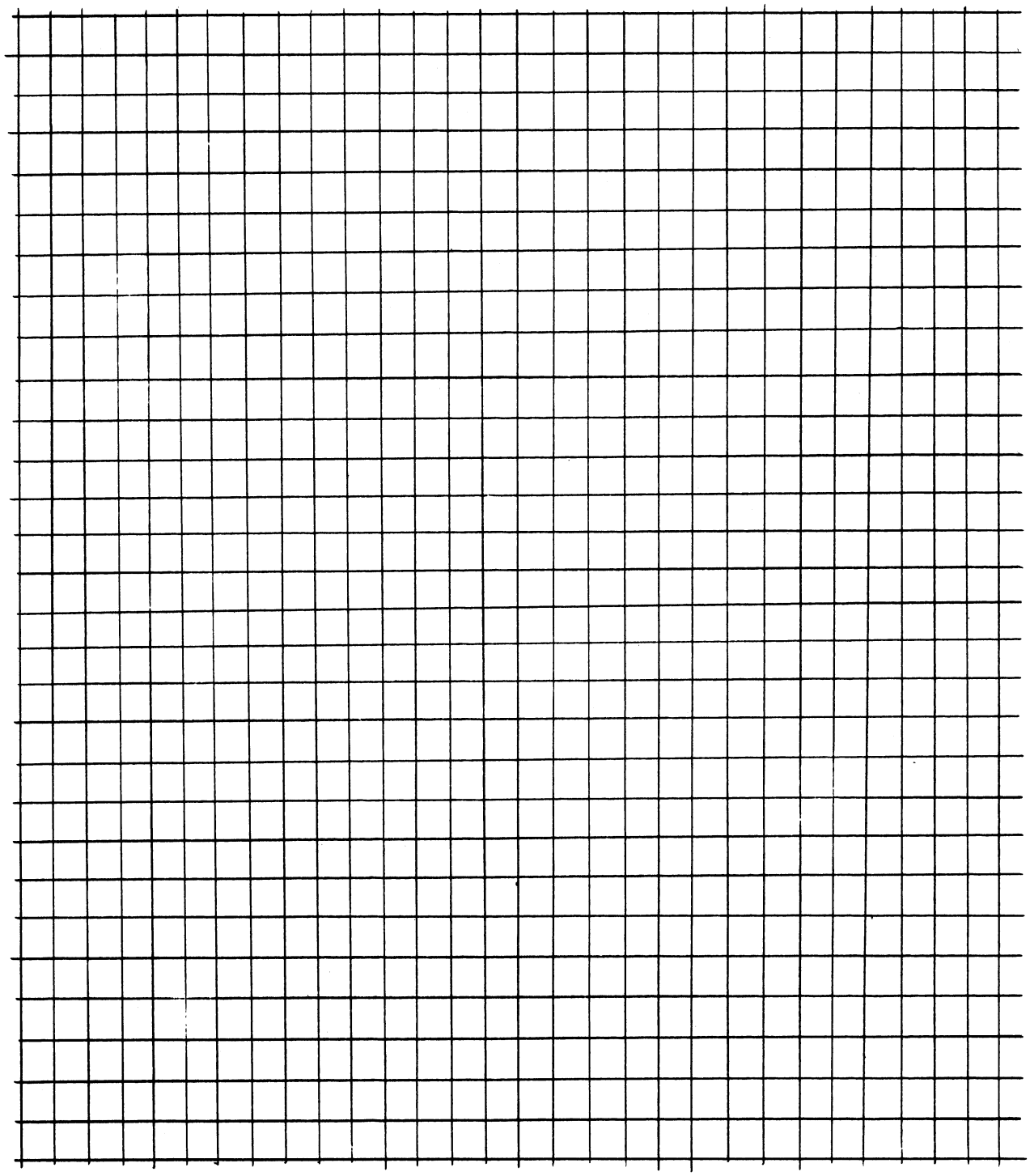


b.


START
Clear AMOUNT
Prepare to cycle through each entry of table MONEY
Is contents of 1st (next) entry of COIN a penny?
YES
NO
Add 1 to AMOUNT
Is contents of 1st (next) entry of COIN a nickel?
YES
NO
Add 5 to AMOUNT
Is contents of 1st (next) entry of COIN a dime?
YES
NO
Add 1¢ to AMOUNT
Is contents of 1st (next) entry of COIN a quarter?
YES
NO
Add 25 to AMOUNT
Add 5¢ to AMOUNT
Are there anymore coins to examine?
NO
YES
STOP

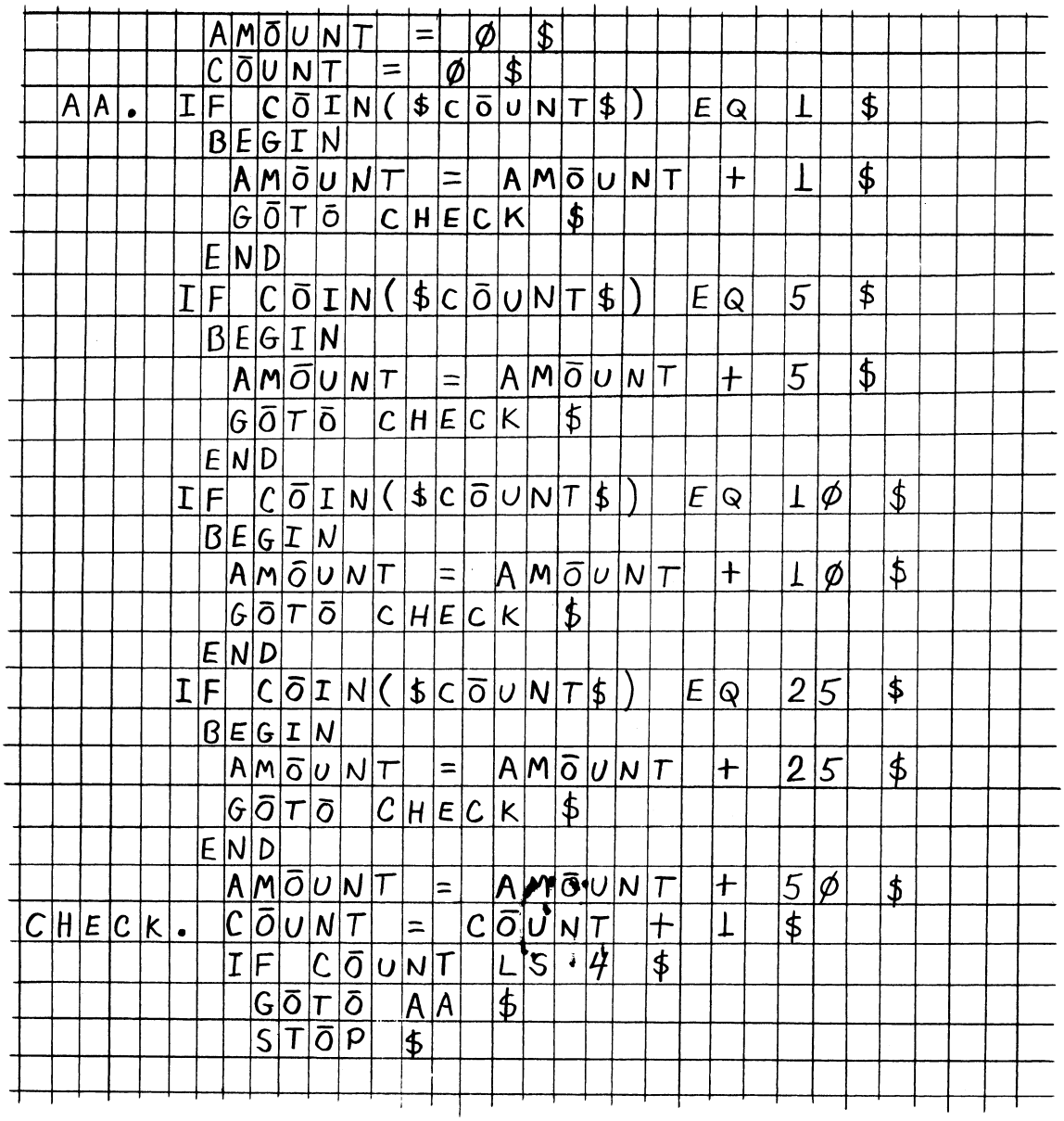
EXERCISES

5.5 Write the JOVIAL coding for exercise 5.4b.

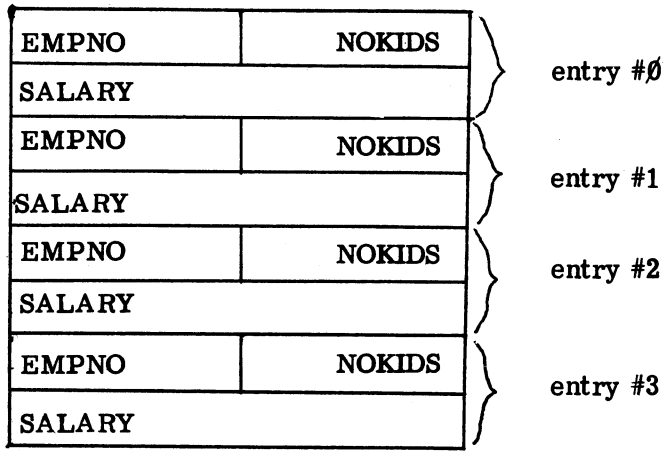


ANSWER TO PRECEDING EXERCISE

5.5



Let us look at another problem involving a table. We will call the table PERØ and have it contain information about employees in a company. Assume that there are four employees and the table appears as follows:



EMPNO is an item that contains a number which identifies a particular employee. It is the "Employee Number."

NOKIDS is an item that contains a number stating how many children an employee has.

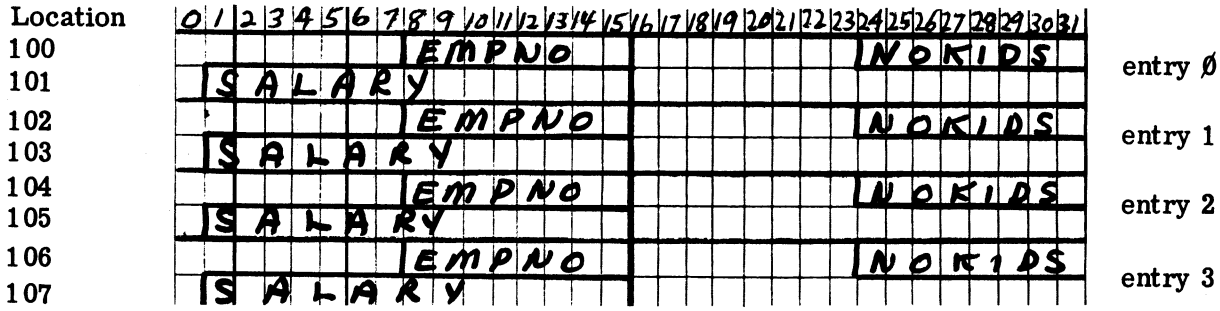
SALARY is an item that contains a number corresponding to an employee's salary in dollars.

Following is the way table PERØ and its items might be defined:

```
TABLE PERØ R 4 $
  BEGIN
    ITEM EMPNO  A  8  U $
    ITEM NOKIDS A  8  U  $
    ITEM SALARY A 15  U  $
  END
```

Let us assume that this table is being defined for use on the AN/FSQ-1 computer which has a core memory word size of 32 bits. Arithmetic restrictions of the Q-7 limit the word length to 15 magnitude bits, or 16 bits including sign for a signed number.

Following is a representation of how PERØ might appear in the AN/FSQ-7 core memory:



The 32 bits are numbered from 0 through 31. Bits 0 and 16 are sign bit positions in the left and right half-words respectively. Each bit position can hold a 1 or a 0. In actuality, each bit position is a tiny donut-shaped piece of iron which can be magnetized in one direction or the other - thus storing a 1 or 0. Each 32 bit slot is referred to as a core location (or core memory register).

I have numbered these at the left, arbitrarily starting with 100. We can say that table PERØ occupies 8 core memory registers. Each entry of PERØ consists of two core memory registers.

When defining table PERØ, I did not specify a starting location. The compiler will therefore establish a starting location for the table.

Also, when defining PERØ, I did not specify into which bit positions to place items EMPNO, NOKIDS, and SALARY. The defining of items for specific bit positions in the core memory register is called packing. Later on, we will see how it is possible for the JOVIAL programmer to specify the packing of items as he wishes. For the present, let us not specify packing and thus the JOVIAL compiler will do the packing of items for us.

Now let us state a very, very important fact. The subscript in JOVIAL pertains to entries and not registers.

Thus, to give the employee in entry #2 a raise in pay of ten dollars, we would write:

```
SALARY ( $ 2 $ ) = SALARY ( $ 2 $ ) + 1Ø $
```

EXERCISES

5.6 The following use table PERØ as described in the text.

a. Write a JOVIAL statement to add one more child to the total number of

children for the employee in entry # 3.

Write a JOVIAL program to compute the total number of children for the employees found in table PERØ. Have this sum placed into item TOTAL. Define TOTAL similar to NOKIDS, however, make it a single item (not in a table).

ANSWERS TO PRECEDING EXERCISES

```
NOKIDS ( $ 3 $ ) = NOKIDS ( $ 3 $ ) + 1 $

ITEM TOTAL A 15 U $  A 15 U $
ITEM COUNT A 15 U $  A 15 U $
TOTAL = Ø $
COUNT = Ø $
TOTAL = TOTAL + NOKIDS ( $ COUNT $ ) $
COUNT = COUNT + 1 $
IF COUNT LS 4 $
GOTO AGAIN $
STOP $
```

EXERCISES

ıe following use table PERØ as described in the text.

Write a program to give a 20 dollar raise to the employee whose employee number matches that of single item RAISE.

Write a program to give a thirty dollar raise to the employee with the most children. (Note: Assume that there is an employee who has the most children).

ANSWERS TO PRECEDING EXERCISES

```
COUNT = Ø $
IF EMPNO ( $ COUNT $ ) EQ RAISE $
BEGIN $
  SALARY ( $ COUNT $ ) = SALARY ( $ COUNT $ ) + 2Ø $
  GOTO EXIT $
END $
  COUNT = COUN T + 1 $
  IF COUNT LS 4 $
  GOTO AGAIN $
  STOP $
```

The following is only one of many possible solutions:

```
ITEM ENTRY  A 15 U $
ITEM EKIDS  A 15 U $
ENTRY = Ø $
```

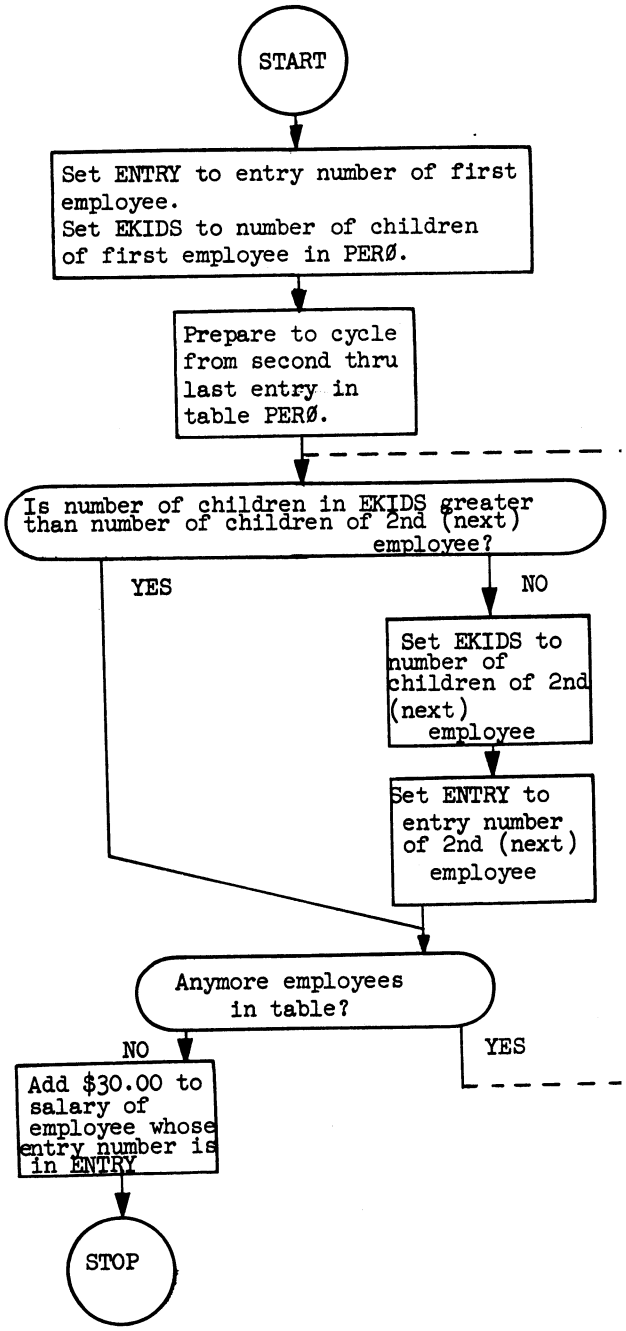
```
          EKIDS = NOKIDS ( $ Ø $ ) $
          COUNT = 1  $
AGAIN.    IF EKIDS GR NOKIDS ( $ COUNT $ )  $
            GOTO CHECK  $
            EKIDS = NOKIDS ( $ COUNT $ )  $
            ENTRY = COUNT$
CHECK.      COUNT = COUNT + 1 $
            IF COUNT LS 4  $
            GOTO AGAIN  $
            SALARY ( $ ENTRY $ ) = SALARY ( $ ENTRY $ ) + 3Ø  $
            STOP $
```

Note: Items EKIDS and ENTRY are just a couple of items that I wish to use for temporary storage.

EXERCISES

5.8 Make a flow diagram for the solution given to problem 5.7b.

ANSWER TO PRECEDING EXERCISE



Any type of variable or arithmetic expression may be used to designate the entry number of an item. When t' e value used to designate the entry number has a fractional part, the fractional bits are truncated, leaving only the integer part.

Examples:

```
ABLE ( $ COUNT + 1 $ )

BAKER ( $ ABLE ( $ DOG * 2 $ ) $ )
```

When the programmer chooses an item or an arithmetic expression to designate the entry number, it is necessary to "manually" increment or decrement this factor and to "manually" test to see if all entries have been processed.

This concludes Chapter 5 which has been concerned with tables, subscripted items, and flow diagramming. In Chapter 6 we will study a new JOVIAL statement called the FOR statement. The FOR statement will simplify greatly the processing of information in tables and the use of subscripted variables.

CHAPTER 6

THE FOR STATEMENT

We have seen in Chapter 5 that the processing of entries in a table can be simplified by using a variable (item) for a subscript.

We recall that to sum the contents of each entry of item NUMBR in 1Ø entry table NUMØ, we were able to write:

```
          ITEM TOTAL  A  16  S  $
          ITEM COUNT  A  15  U  $
          TOTAL = Ø  $
          COUNT = Ø  $
AGAIN.    TOTAL = TOTAL + NUMBR ( $ COUNT $  )  $
          COUNT = COUNT + 1  $
          IF COUNT LS 1Ø  $
          GOTO AGAIN $
          STOP $
```

In the program above, the single item COUNT took on all of the values from Ø through 9, thus obtaining the sum of each entry of item NUMBR in the ten entry table.

It is possible, however, to define a variable subscript in JOVIAL in an easier manner than by the usual item definition, which we used with COUNT.

This easy definition of a variable subcript can be achieved by use of the FOR statement.

Our program would appear as follows:

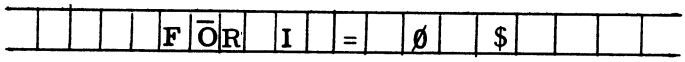
```
          ITEM TOTAL  A  16  S  $
          TOTAL = Ø  $
          FOR I = Ø  $
          BEGIN  $
AGAIN.      TOTAL = TOTAL + NUMBR ( $ I  $ )  $
            I = I + 1  $
            IF I LS 1Ø  $
            GOTO AGAIN  $
            STOP $
          END  $
```

The FOR in the FOR statement is a code word that lets the compiler know that we want to define a variable subscript. The FOR is a code word just as ITEM was a code word in an item definition. However, FOR is a dynamic JOVIAL program statement and, thus part of the program and not part of the definitions.

Following the code word FOR comes a space and then the name that we wish to give the subscript. In the example, we have given the subscript the name I. Any letter in the

alphabet may be used. The subscript name always consists of a single letter.

Then follows an equal sign, "=", and then the initial value which we wish the subscript to have. Finally, comes the dollar sign.



No spaces need surround the "=" sign or need to precede the dollar sign.

The statement FOR I = Ø has taken the place of the two statements:

```
ITEM COUNT  $ A  15  U  $
COUNT = Ø $
```

The FOR statement as we have used it defines the variable subscript and gives it an initial setting.

The FOR statement, however, defines a subscript for only the first statement following the definition. That is why I made a compound statement out of the several statements following the FOR statement in the last example.

The BEGIN and END brackets make the several statements into a compound statement and thus, the subscript I is defined for the entire compound statement.

Within the program area for which the subscript is defined, it can be used as a subscript attached to an item name as in:

```
TOTAL = TOTAL * +  NUMBR ( $ I $ ) $
```

In the above statement, the subscript use requires the symbology of ($ $) as was used previously with item COUNT.

Also within the program area for which the subscript is defined, it can be used in the same manner as item COUNT was, i.e., just by referring to it by name. Thus,

```
I = I + 1 $
IF I LS 1Ø $
```

In the above two statements, the symbology of ($ $) is not used.

EXERCISES

6.1 a. Given the following FOR statement

```
FOR N = 3 $
```

(1) What is the code word that tells the compiler that we want to define a subscript?

(2) What name is being given to the subscript?

(3) What initial value will be placed into the subscript?

For how many statements following the FOR statement is the subscript defined?

How can the subscript definition created by a FOR statement be made valid for several JOVIAL statements?

Write a JOVIAL statement which will define a subscript called P and which will set P initially to the value 5.

What is wrong with the following JOVIAL program?

```
FOR J = 9 $
IF NUMBR ( $ J $ ) GR 39 $
  NUMBR ( $ J $ ) = Ø $
  J = J -1 $
  IF J GQ Ø $
    GOTO AGAIN  $
    STOP $
```

Write a JOVIAL program using a subscript called V which will keep a count in item COUNT of all entries of item NUMBR in 1Ø entry table NUMØ, which contain values between 79 and 84 inclusive.

ANSWERS TO PRECEDING EXERCISES

(1) The code word is FOR.

(2) The name being given to the subscript is N.

(3) The initial value placed into the subscript is 3.

The FOR statement defines a subscript for only the first statement following it. (Actually, to be completely correct - it applies to the first non-FOR statement following it. This is the case where we have two or more FOR statements consecutively where we wish to define two or more subscripts. More on this in a succeeding chapter).

Note, however, that a FOR statement may be made to apply to several JOVIAL statements by incorporating the several statements within BEGIN and END brackets, thus creating one compound statement.

A subscript definition may be made to apply to several JOVIAL statements by making a compound statement out of the several statements.

```
FOR P = 5  $
```

e. The subscript J is undefined for all of the statements except the IF statement. Therefore, references to subscript J in

```
NUMBR ( $ J$  ) = Ø  $
J = J -1  $
IF J GQ Ø  $
```

are illegal. This program would not get past the compiler. The compiler would produce a printout stating that J is undefined in the aforementioned statements.

The program can be corrected easily by using the brackets BEGIN and END.

f.
```
   COUNT = Ø $
   FOR V = Ø $
     BEGIN $
AGAIN.   IF NUMBR ( $ V $ ) GQ 79 $
           BEGIN
             IF NUMBR ( $ V $ ) LQ 84 $
               COUNT = COUNT + 1 $
           END
           V = V  + 1 $
           IF V LS 1Ø $
             GOTO AGAIN $
             STOP $
     END
```

There are three forms for the FOR statement. They are:

The Incomplete FOR statement
The Partial FOR statement
The Complete FOR statement

The FOR statement which we have already discussed, was the incomplete FOR statement. Use of the incomplete FOR statement permits us to automatically establish an initial value for the subscript.

Use of the partial FOR statement permits us to automatically establish an initial value and to automatically increment the subscript by an increment factor.

Use of the complete FOR statement permits us to automatically establish an initial value, automatically increment the subscript, and automatically test the subscript.

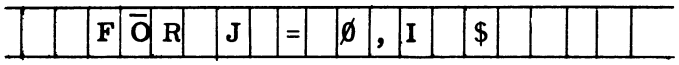
Examples of each are:

The Incomplete FOR Statement

		F	O	R		I		=		Ø		$							

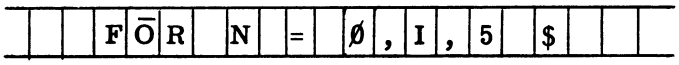
This defines a subscript called I and sets it initially to zero.

The Partial FOR Statement



This defines a subscript called J, sets it initially to zero, and results in automatically incrementing I by 1 on each succeeding pass through the loop associated with the subscript I.

The Complete FOR Statement



This defines a subscript called N, sets it initially to zero, results in automatically incrementing N by 1 on each pass, and tests N on each pass of the associated loop to see if it has reached a value greater than 5. If N has become greater than 5, the looping is not continued.

Let us now sum the contents of each entry of item NUMBR in ten entry table NUMØ, using the partial FOR statement.

```
TOTAL = Ø $
FOR B = Ø, 1 $
  BEGIN
    TOTAL = TOTAL + NUMBR ( $ B $ ) $
    IF B GQ 1Ø $
      STOP $
END
```

Here we see that the subscript B is initially set to zero. Then the contents of entry #Ø of item NUMBR is added to TOTAL. Since on the first ten passes, the subscript B will contain less than 1Ø (B will contain Ø, 1, 2, 3, 4, 5, 6, 7, 8, and 9 respectively on the first ten passes) the IF statement will be false and control will go to END.

It is the END that will result in automatically incrementing subscript B by the increment factor of 1 and transferring control back to BEGIN.

We do not see any "incrementing" and "transferring" associated with the END but we must realize that the compiler will create these functions for us in the intermediate level language, which it sets up during compilation (the changing of our JOVIAL program to binary).

Thus, the END of a compound statement which is associated with a subscript appears on JOVIAL coding paper exactly like the END of a compound statement, which is associated with an IF statement. However, with a partial FOR statement, the compiler will produce symbolic machine instructions to increment and transfer for the former, but not the latter.

It would be well to state that END cannot be labeled even through BEGIN can. Later we will see how we can go directly to an END which is associated with a subscript from elsewhere in the program.

EXERCISES

6.2 Given item NUMBR in ten entry table NUMØ as described in the text and also given item TALLY defined as ITEM TALLY A 15 U, answer the following questions:

a. What information will TALLY contain after the following program has operated?

```
TALLY = Ø $
FOR K 1 Ø, 1 $
BEGIN
  IF NUMBR ( $ K$ ) EQ 6 $
    TALLY = TALLY + 1 $
    IF K GQ 1Ø $
    STOP $
END
```

b. Write a program, using a partial FOR statement, that will obtain the sum of the positive numbers in item NUMBR. Have the sum placed in TALLY. Have your program set all other entries of item NUMBR to 99.

c. Write a program, using a partial FOR statement, that will square the contents of every other entry of item NUMBR. Start with the second entry (entry #1).

ANSWERS TO PRECEDING EXERCISES

6.2 a. TALLY will contain a count of the number of entries of item NUMBR which contained the value 6.

b.
```
TALLY = Ø $
FOR L = Ø, 1 $
BEGIN
  IF NUMBR ( $ L $ ) GR Ø$
    BEGIN
      TALLY = TALLY + NUMBR ( $ L $ ) $
      GOTO CHECK $
    END
      NUMBR ( $ L $ ) = 99 $
CHECK. IF L GQ 1Ø $
        STOP $
END
```

An alternate solution follows:

```
TALLY = Ø $
FOR P = Ø, 1 $
BEGIN
  IF NUMBR ( $ P $ ) GR Ø $
  TALLY = TALLY + NUMBR ( $ P $ ) $
  IF NUMBR ( $ P $ ) LS Ø $
  NUMBR ( $ P $ ) = 99 $
  IF L GQ 1Ø $
  STOP $
END
```

c.
```
FOR P = 1, 2  $
BEGIN
  NUMBR ( $P$ ) NUMBR ( $P $ ) * NUMBR ( $ P $ ) $
  IF P GQ 1Ø $
  STOP $
END
```

Note that in the above solution P will take on the values 1, 3, 5, 7, and 9 before the program stops.

We have seen how we can sum the contents of each entry of item NUMBR using first, an incomplete FOR statement, and second, a partial FOR statement.

Let us now write the program using the complete FOR statement.

```
TOTAL = Ø $
FOR Z = Ø, 1, 9 $
TOTAL = TOTAL + NUMBR ( $ Z $ ) $
STOP $
```

Here, the first statement following the FOR statement is performed as many times as is necessary to have the subscript Z start with an initial value of Ø, be incremented by 1 each time, until Z has a final value of 9.

Thus, the statement TOTAL = TOTAL + NUMBR ($ Z $) $ will be performed ten times with the value of Z ranging from Ø through 9.

Since only one statement is involved in the loop of the above example, the statement was not a compound statement and therefore, no BEGIN and END brackets were required. Usually however, the loop will consist of several statements and will be a compound statement.

EXERCISES

6.3 Write a JOVIAL program which will add 5 to every entry of item NUMBR which contains 29 or greater. Otherwise, set the entry of NUMBR to 7. Use the complete FOR statement.

6.3

```
      FOR M = Ø, 1, 9 $
      BEGIN
        IF NUMBR ( $ M $ ) GQ 29 $
          BEGIN
          NUMBR ( $ M $ ) = NUMBR ( $ M $ ) + 5 $
          GOTO CHECK $
          END
            NUMBR ( $ M $ ) = 7 $
CHECK.    NUMBR ( $ M $ ) = NUMBR ( $ M $ ) $
    END
    STOP $
```

In order to get to the END from the true portion of the IF statement, I put in a dummy statement so that I could label it CHECK. In the next chapter, we will obtain additional JOVIAL capability (by use of an operator called TEST) and a dummy statement with a label will not be necessary to get to the subscript END.

A second solution follows:

```
FOR M = Ø, 1, 9 $
BEGIN
  NUMBR ($ M $ ) = NUMBR ( $ M $ ) + 5 $
  IF NUMBR ( $ M $ ) LS 34 $
    NUMBR ( $ M $ ) = 7 $
  END
  STOP $
```

A third solution follows:

```
FOR M = Ø 1, 9 $
BEGIN
  IF NUMBR ( $ M $ ) GQ 29 $
     NUMBR ( $ M $ ) = NUMBR ( $ M $ ) + 5 $
  IF NUMBR ( $ M $ ) LS 29 $
     NUMBR ( $ M $ ) = 7 $
END
STOP $
```

Let us summarize the different parts of the FOR statement. The general format for the complete FOR statement is:

```
FOR L = A, B, C $
```

L in the above format may be any single alphabetic character from A to Z. Subsequent references in the program to the subscript will use this letter.

A in the above format functions as the initial value of the subscript.

B in the above format functions as the increment or decrement (the B factor may be negative). The B value is added to or subtracted from the current value of the subscript L to determine the next value of L.

C in the above format functions as the test factor in the control of the loop. The value of C designates the final value of L.

Any or all of the factors A, B, and C may contain an item of any type, a subscript which has an initial value, a constant, or any legal arithmetic expression. The value of the A and C factors should always be positive.

Examples:

```
FOR I  = ABLE, I/2, K*5 $

FOR J  = Ø, 1Ø, 99 $

FOR K  = 9, -1, Ø $

FOR R  = J, J-1, J+9 $
```

The B and C factors of a complete FOR statement are checked at the end of each iteration of the loop, therefore, the value of either of these factors may be changed during the cycle.

At the end of each pass through a loop the value of the B factor is added to the current value of the subscript to obtain the next value of the subscript.

The a test is automatically made in the following way to determine if the iterations of the loop have been completed:

First - If the B factor of the FOR statement is a positive value, a comparison is made to see if the current value of the subscript is greater than the value of the C factor. If the B factor of the FOR statement is a negative value, a comparison is made to see if the current value of the subscript is less than the value of the C factor.

Second - If the comparison of subscript with C factor is greater (for positive B factor) or less (for negative B factor) control is not returned to the beginning of the loop (the first statement following FOR statement if that statement is not another FOR statement). Instead, control proceeds with the statement following the loop.

Finally, let us mention in passing, that the subscript need not be referenced in the loop. The following is a valid program:

```
FOR K = Ø, 1, 5 $

BEGIN

  SUM = SUM + 1 $

  COUNT = COUNT + 2 $

END
```

In the above program the subscript K is not referenced within the loop. The subscript K merely results in 6 passes through the loop.

This concludes Chapter 6.

CHAPTER 7

STATUS ITEMS, HOLLERITH ITEMS, THE EXCHANGE STATEMENT, AND THE TEST STATEMENT

The only type of item that we have defined and used up to this point has been the integer or A-type of item. Now let us introduce a new type called the status or S-type.

Use of the status type item enables the programmer to program in a more symbolic manner than otherwise might be possible.

Suppose, for example, we wished to have an item called COLOR which would contain one of several possible values. The possible values might be RED, BLUE, GREEN, WHITE, AND BLACK.

With the capability we have up to this point we could logically code a three-bit integer type item called COLOR so that the following relationships were understood:

Binary Number	Logical Equivalent
ØØØ	RED
ØØ1	BLUE
Ø1Ø	GREEN
Ø11	WHITE
1ØØ	BLACK

Then if we wanted to set item COLOR to WHITE we could write:

```
COLOR = 3$
```

If we wanted to determine if item COLOR contained BLUE we could write:

```
IF COLOR EQ 1$
```

However, if we define item COLOR as a status type item rather than an integer type item we can set it to WHITE by the JOVIAL statement:

```
COLOR = V (WHITE)$
```

We can determine if item COLOR contains the value for BLUE by using the JOVIAL statement:

```
IF COLOR EQ V (BLUE) $
```

The symbology of "V( )" must be used when writing a status constant.

Thus, the program will be written in a manner approaching the ideal of pure English and mathematics.

We can define the status item COLOR as we have discussed by using the following item definition:

```
ITEM COLOR S V(RED) V(BLUE) V(GREEN)
           V(WHITE) V(BLACK) $
```

First, we write the code word ITEM followed by the name which we wish to give the item. As with an integer type item, the name consists of two or more alphanumeric symbols with the first character always a letter.

Then S follows, which indicates that the item will contain a status value, i.e., each value of this item stands for a unique status.

Finally, every possible status is listed, given in one or more alphanumeric characters, the first character being a letter. A status item must have at least two statuses.

Each status must be enclosed in parentheses and preceded by the letter V. No two statuses may be represented by the same arrangement of characters (i.e., if one status in the item definition is ABC, another in the same item definition cannot be ABC also; but may be BCA).

Two separate status items, however, may have statuses of identical character representations.

The following may be used in the same program:

```
ITEM ALPHA S V (YES) V (NO) $

ITEM ALARM S V (YES) V (NO) V (MAYBE) $

ITEM SOLD S V (NO) V (YES) $
```

EXERCISES

7.1 a. Define a status item called DAY which will have the possible statuses of MON, TUES, WED, THUR, FRI, SAT, and SUN.

b. Write a JOVIAL statement that will set a status item called COLOR to a status of RED.

c. What are the contents of items MARY and JANE after the following program has operated?

```
MARY = V (YES) $

JANE  = V (NO) $

IF JANE EQ V (MAYBE) $

  BEGIN

     MARY = JANE $

     JANE  = V (NO) $

     GOTO EXIT  $

  END

     JANE  = MARY  $

     MARY = V (MAYBE)  $

EXIT.   STOP $
```

d. Given the following item and table definitions:

```
TABLE PERØ R 1ØØ $

  BEGIN

     ITEM EMPNO A 8 U $

     ITEM SALARY A 15 U $

     ITEM SEX S V (MALE) V (FEMALE) $

     ITEM MARIED S V (NO) V (YES) $

  END

     ITEM COUNT A 1Ø U $
```

Make a flow diagram and write a program which will count in item COUNT how many employees are unmarried women who have a salary of $550.00 or more.

ANSWERS TO PRECEDING EXERCISES

7.1 a.

```
ITEM DAY S V(MON) V(TUES) V(WED)
V(THUR) V(FRI) V(SAT) V(SUN) $
```

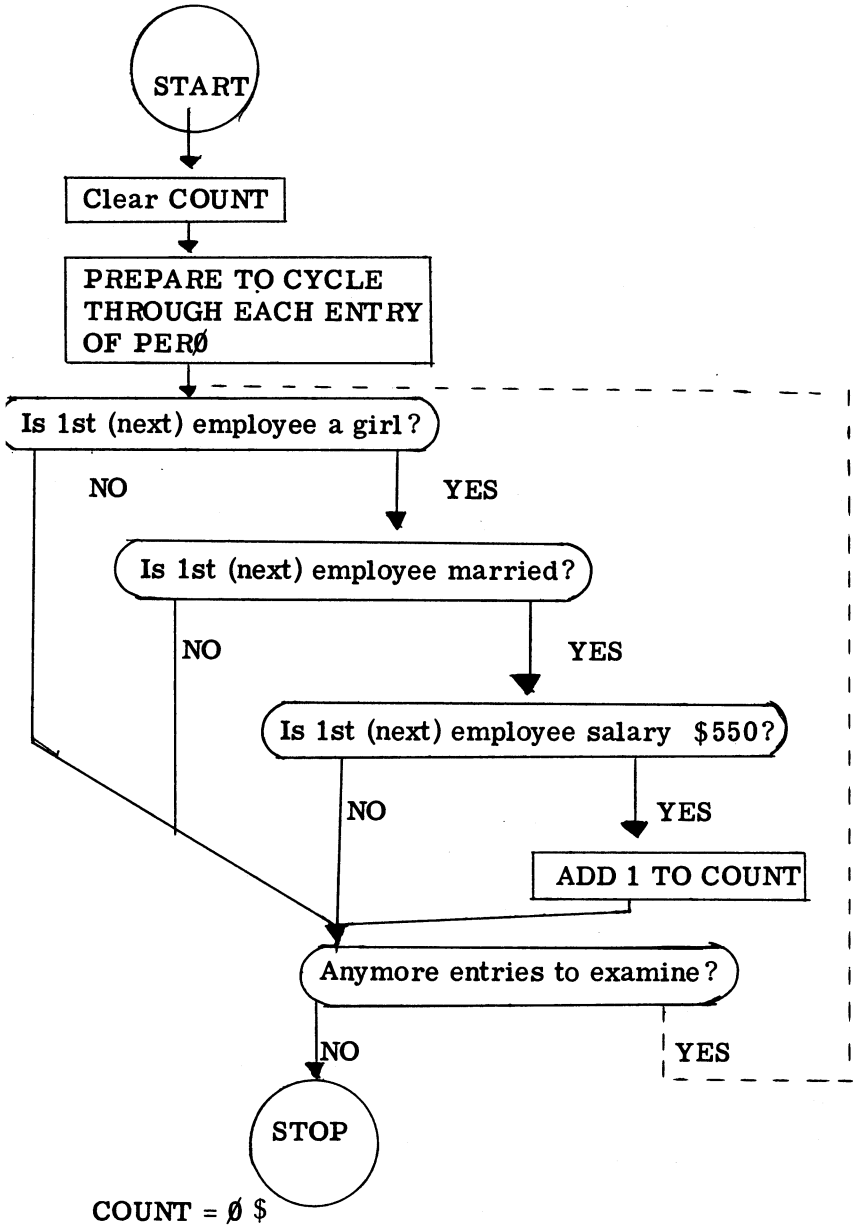
b.

C Ō L Ō R = V (R E D) $

c. MARY contains a status of MAYBE

JANE contains a status of YES

d.



COUNT = Ø $
FOR A = Ø, 1, 99$
BEGIN

d. (Continued)

```
IF SEX ( $A$  ) EQ V (FEMALE) $
  BEGIN
    IF MARIED ( $ A$ ) EQ V (NO) $
      BEGIN
        IF SALARY ( $A $ ) GQ  55Ø $
          COUNT = COUNT + 1 $
      END
    END
  END
  STOP $
```

An alternate solution follows:

```
COUNT = Ø $
FOR A = 0, 1, 99 $
BEGIN
  IF SEX ( $ A $)EQ V (MALE)  $
    GOTO CHECK $
  IF MARIED ( $A$ ) EQ V (YES) $
    GOTO CHECK $
  IF SALARY ( $A$ ) GQ 55Ø $
    COUNT = COUNT + 1 $
  CHECK. COUNT = COUNT $
  END
  STOP $
```

In the second solution given for exercise 7.1d it was desirable to get to the END that was associated with the A subscript loop. For this purpose the dummy statement CHECK. COUNT = COUNT $ was included since END cannot be labeled. Then a GOTO CHECK $ in effect transferred control to END.

This awkwardness can be eliminated by using the JOVIAL statement called TEST.

The solution would then be written as follows:

```
COUNT =  Ø $
FOR A =  Ø, 1, 99$
BEGIN
  IF SEX ( $ A$ ) EQ V (MALE) $
    TEST $
  IF MARIED ( $ A $ ) EQ V (YES) $
    TEST $
  IF SALARY ( $A$ ) GQ 55Ø $
    COUNT = COUNT + 1 $
END
  STOP $
```

In the above solution, the statement TEST simply causes control to be transferred to the

END which is the end of the subscript loop (where incrementing and testing of the subscript will take place).

There is an additional feature of the TEST statement that will add to its flexibility, however, we will save that discussion for a later time.

EXERCISES

7.2 a. What is the content of item SUM after the following program has operated?

```
ITEM SUM A 15 U $
SUM = 1Ø$
FOR I = Ø, 1, 9 $
BEGIN
  IF I LQ 5 $
    BEGIN
      SUM = SUM-1 $
      TEST $
    END
      SUM = SUM + 1 $
  END
    STOP $
```

b. Given the following item and table definitions

```
TABLE PERØ R 1ØØ $
  BEGIN
    ITEM EMPNO A  8  U$→A 8 U$
    ITEM SALARY A 15 U $→A 15 U $
    ITEM SEX  S V (MALE) V (FEMALE)   $
    ITEM MARIED S V (NO) V (YES) $
  END
    ITEM NEWWED A  8  U  $→A  8  U  $
```

Write a program using the TEST statement that will give a $10.00 raise to the employee whose EMPNO matches the contents of single item NEWWED. Also set the item MARIED of the matched employee to a status of YES.

ANSWERS TO PRECEDING EXERCISES

7.2 a. SUM will contain 8 after the program has operated. This is due to the fact that the subscript I will range from Ø through 9 and six of these values of I will be less than or equal to 5; namely, Ø, 1, 2, 3, 4, and 5, thus a value of 6 is subtracted from SUM to give 4. Then 4 will be added to SUM since four values of I will be greater than 5; namely, 6, 7, 8, and 9.

b.
```
FOR J = Ø, 1, 99 $
  BEGIN
    IF EMPNO ( $J $) NQ NEWWED $
      TEST$
      SALARY ( $J $) = SALARY ( $J$ ) + 1Ø$
```

7.2 b. (Continued)

```
    MARIED ( $J $) = V (YES) $
    STOP $
END
```

Now let us introduce the Exchange Statement. The following is an example of the Exchange Statement:

```
 ALPHA == BETA $
```

The Exchange Statement is characterized by the double equal sign "==" and acts like a two-way assignment statement.

In the above example, the contents of item ALPHA are exchanged with the contents of item BETA.

Both the left term and right term of an exchange statement must be variables (items, subscripts, etc.). They cannot be constants or arithmetic expressions.

Note that the exchange symbol (==) is coded as two assignment statement symbols with no space between.

EXERCISES

7.3 a. Write a JOVIAL statement that will exchange the contents of items RHO and GAMMA.

b. What are the contents of items ALPHA and BETA after the following program operates?

```
ALPHA = 6 $
BETA = 9 $
IF ALPHA GR BETA -4 $
  ALPHA == BETA $
  BETA == ALPHA $
  STOP $
```

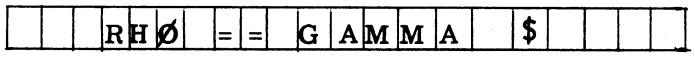
c. Given the following table and item definitions

```
TABLE NUMØ  R  5  $
  BEGIN
    ITEM NUMBR A 15 U  $
  END
```

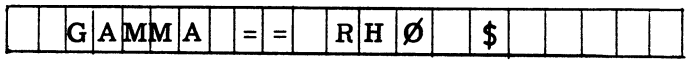
Write a program using the exchange statement that will sort the contents of table NUMØ into ascending order; i.e., smallest number in entry #Ø and largest in entry #4.

ANSWERS TO PRECEDING EXERCISES

7.3 a.



or



b. ALPHA will contain 6 and BETA will contain 9 after the program has operated. This is because the IF statement will be true and thus both exchange statements will be performed. Incidentally, even though the two exchange statements are written in reverse order, they both have the same result - namely, the exchange of the contents of items ALPHA and BETA.

c.
```
AGAIN.  FOR I = Ø, 1, 3 $
        BEGIN
          IF NUMBR ($ I$ ) LQ NUMBR ($I + 1$ ) $
          TEST $
          NUMBR ($ I$ ) == NUMBR ($ I + 1$ ) $
          GOTO AGAIN $
        END
          STOP $
```

Note that since I am using item NUMBR subscripted by I + 1, I want subscript I to run only as high as 3. This will then result in I + 1 having a value of 4 which is the last entry in a 5 entry table.

Another type of item which is popular in JOVIAL programming is the hollerith or H-type item.

Information in a hollerith type item is coded to represent alphanumeric and punctuation-like characters. The following six-bit hollerith code is employed:

6-Bit Hollerith Code

Character	6-Bit Representation	Character	6-Bit Representation
0	ØØØØØØ	A	Ø1ØØØ1
1	ØØØØØ1	B	Ø1ØØ1Ø
2	ØØØ1Ø	C	Ø1ØØ11
3	ØØØ11	D	Ø1Ø1ØØ

6-Bit Hollerith Code (Continued):

Character	6-Bit Representation	Character	6-Bit Representation
4	ØØØ1ØØ	E	Ø1Ø1Ø1
5	ØØØ1Ø1	F	Ø1Ø11Ø
6	ØØØ11Ø	G	Ø1Ø111
7	ØØØ111	H	Ø11ØØØ
8	ØØ1ØØØ	I	Ø11ØØ1
9	ØØ1ØØ1	J	1ØØØØ1
=	ØØ1Ø11	K	1ØØØ1Ø
,	111Ø11	L	1ØØØ11
$	1Ø1Ø11	M	1ØØ1ØØ
.	Ø11Ø11	N	1ØØ1Ø1
+	Ø1ØØØØ	O	1ØØ11Ø
-	1ØØØØØ	P	1ØØ111
*	1Ø11ØØ	Q	1Ø1ØØØ
/	11ØØØ1	R	1Ø1ØØ1
'	ØØ11ØØ	S	11ØØ1Ø
(	1111ØØ	T	11ØØ11
)	Ø111ØØ	U	11Ø1ØØ
Blank	11ØØØØ	V	11Ø1Ø1
		W	11Ø11Ø
		X	11Ø111
		Y	111ØØØ
		Z	111ØØ1

Therefore, if an item called NAME were to contain the name ESTHER it would be necessary for the item to consist of at least 36 bits. We would define the item as a hollerith or H-type as follows:

	I	T	E	M		N	A	M	E		H		6		$	

After the code word ITEM, comes the unique name to be given to the item - in this case, NAME. Then follows an H to indicate that we are defining a hollerith type item. Next follows a digit which specifies the number of hollerith characters involved - in this example, 6.

The Q-7 computer word is 32 bits in length and can hold up to 5 hollerith characters per word with 2 bits left over. Hollerith (lateral) items and constants greater than 5 characters will occupy consecutive computer locations and must begin in bit Ø of the first word.

To place the name ESTHER into the hollerith item NAME we could use the following assignment statement:

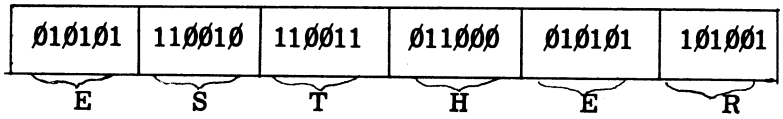
NAME = 6H (ESTHER) $

The actual hollerith constant is enclosed in parentheses and preceded by a digit signifying

the number of hollerith characters involved and then the letter "H". Note that a blank is a legal hollerith character.

The item called NAME in the preceding example would appear as follows after the given assignment statement had been performed.

Item NAME



We use the hollerith constant in a similar fashion when testing with an IF statement.

IF AIRCFT EQ 6H (KC-135) $

Note: If the number preceding the H for a constant is less than the number preceding the H in the item definition, leading zeros will be inserted.

Item Definition: ITEM ALPHA H 4 $

Assignment Statement: ALPHA = 3H (ABC) $

Result:

Item ALPHA

000000	010001	010010	010011

EXERCISES

7.4 a. Define a hollerith type item called MONTH and make it capable of holding 8 hollerith characters.

b. Write the assignment statement that will set item MONTH, which you defined in part a above, to the 8-character constant of DECEMBER.

c. Show the item MONTH in binary as it would appear after the assignment statement of part b above is performed.

d. Write the assignment statement that will set item MONTH, which you defined in part a above, to the constant of APRIL.

e. Show the item MONTH in binary as it would appear after part d above.

f. Write an IF statement that will determine if the value in item MONTH is AUGUST.

ANSWERS TO PRECEDING EXERCISES

a. ITEM MONTH H 8 $

b. MONTH = 8H (DECEMBER) $

c.

Ø1Ø1ØØ	Ø1Ø1Ø1	Ø1ØØ11	Ø1Ø1Ø1	1ØØ1ØØ	Ø1ØØ1Ø	Ø1Ø1Ø1	1Ø1ØØ1
D	E	C	E	M	B	E	R

d. MONTH = 5H (APRIL) $

e.

ØØØØØØ	ØØØØØØ	ØØØØØØ	Ø1ØØØ1	1ØØ111	1Ø1ØØ1	Ø11ØØ1	1ØØØ11
Ø	Ø	Ø	A	P	R	I	L

f. IF MONTH EQ 6H (AUGUST) $

SAVE A LIFE

If you observe an accident involving electrical shock,
DON'T JUST STAND THERE - DO SOMETHING!

RESCUE OF SHOCK VICTIM

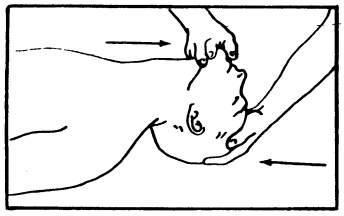
The victim of electrical shock is dependent upon you to give him prompt first aid. Observe these precautions:

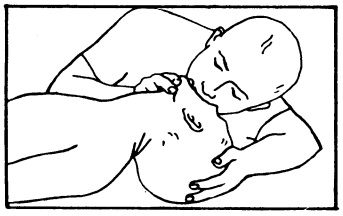
1. Shut off the high voltage.
2. If the high voltage cannot be turned off without delay, free the victim from the live conductor. REMEMBER:
   a. Protect yourself with dry insulating material.
   b. Use a dry board, your belt, dry clothing, or other non-conducting material to free the victim. When possible PUSH - DO NOT PULL the victim free of the high voltage source.
   c. DO NOT touch the victim with your bare hands until the high voltage circuit is broken.

FIRST AID

The two most likely results of electrical shock are: bodily injury from falling, and cessation of breathing. While doctors and pulmotors are being sent for, DO THESE THINGS:

1. Control bleeding by use of pressure or a tourniquet.
2. Begin IMMEDIATELY to use artificial respiration if the victim is not breathing or is breathing poorly:
   a. Turn the victim on his back.
   b. Clean the mouth, nose, and throat. (If they appear clean, start artificial respiration immediately. If foreign matter is present, wipe it away quickly with a cloth or your fingers).





   c. Place the victim's head in the "sword-swallowing" position. (Place the head as far back as possible so that the front of the neck is stretched).
   d. Hold the lower jaw up. (Insert your thumb between the victim's teeth at the midline - pull the lower jaw forcefully outward so that the lower teeth are further forward than the upper teeth. Hold the jaw in this position as long as the victim is unconscious).
   e. Close the victim's nose. (Compress the nose between your thumb and forefinger).
   f. Blow air into the victim's lungs. (Take a deep breath and cover the victim's open mouth with your open mouth, making the contact air-tight. Blow until the chest rises. If the chest does not rise when you blow, improve the position of the victim's air passageway, and blow more forcefully. Blow <u>forcefully</u> into <u>adults</u>, and <u>gently</u> into <u>children</u>.
   g. Let air out of the victim's lungs. (After the chest rises, quickly separate lip contact with the victim allowing him to exhale).
   h. Repeat steps f. and g. at the rate of 12 to 20 times per minute. Continue rhythmically without interruption until the victim starts breathing or is pronounced dead. (A smooth rhythm is desirable, but split-second timing is not essential).

DON'T JUST STAND THERE - DO SOMETHING!