5.1 Introduction

A flowchart is a diagram consisting of labeled geometrical symbols together with rows connecting one symbol to another. It gives a pictorial representation of a data processing procedure.

System Flowcharts

A system flowchart shows the path taken as data pass from one organizational unit or processing cycle may be considered a system flowchart.

Program Flowcharts

A program flowchart pictures the sequence of instructions for solving a particular problem (usually by means of a computer program).

Example 5.1 The Croesus company plans to give a year end 3% bonus to each of its employees. A step-by-step list of instructions for calculating the BOUNDS for a given employee follows:

Step 1. Obtain employee's yearly SALARY from employee's record.
Step 2. Calculate BOUNDS as:

\[ \text{BOUNS} = 0.03 \times \text{SALARY} \]  
Where "\times" means multiply.

Step 3. Record employee's BOUNDS in the bonus file. Figure 5.1 is a program flowchart of the above instructions.

The sequence of instructions for solving a particular problem is called algorithm. Actually whenever an algorithm is much more complicated than that of example 5.1, one usually displays the algorithm through a flowchart rather than through a numbered list instructions.
When used with computer programs, flowcharts offer the following advantages:

1) It is usually much easier to draw a flowchart of an algorithm and then to write the program, than to write the program directly.

2) Flowcharts are an important aid in the development of the algorithm itself.

3) Flowcharts are easier to understand than a program itself; hence computer programs supplied by the manufacturer normally come with their flowcharts.

4) Flowcharts are independent of any particular programming language, so that a given flowchart can be used to transfer an algorithm into more than one programming language.

This chapter is mainly devoted to the study of program flowcharts, so "flowchart" shall mean "program flowchart" unless otherwise stated or implied.

5.2 VARIABLES, DATA- NAMES, PROGRAMMING STATEMENTS

By a variable in a computer program, we mean a data item whose value may change during the execution of the program. A data-name (or variable-name) is a programmer-supplied name or label of a variable. The rules governing these names depend upon the programming language being used. Generally speaking, name consist of a number of alphanumeric characters, with the first character being a letter.

Normally, one chooses names so that they indicate the kind of data item they present; e.g. BONUS might be chosen to denote the bonus of an employee in a company. Observe that the value of BONUS may change during the execution of the program; it may be $300, $400, …, depending on the particular employee. Each programming language has certain "reserved words" such as (FORTRAN's) STOP, END, DO, IF, that have specific operational meaning in the language. These words may not be chosen as variable-names.

During the compiling of a computer program, the variables in the program are assigned specific storage locations in the memory of the computer. We identify a variable with its storage location, e.g. we may speak of "the variable BONUS" or, equivalently, "the storage location named BONUS" or "the address BONUS". The value of a variable at any time is the value then stored at the address of the variable.

Assignment Statements:

A programming statement that assigns a value to a variable, by storing that value at the address of the variable, is called an assignment statement. Recall that the previous content of the storage location is thereby obliterated.

**Example 5.2** Suppose that we want to assign the numerical value 0.08 to a variable (or memory location) called RATE. The assignment statement is:

```
RATE = 0.08
```

Similarly, we would write the assignment statement

```
BONUS = 0.05 * SALARY
```

If the variable BONUS is to be assigned the number obtained by multiplying SALARY by 0.05, where SALARY is a variable that has already been defined, i.e. already has been given a value.

Many high languages e.g., FORTRAN, BASIC, C use the equals sign as above, to indicate that the variable on the left is to be assigned the value of the expression on the right. APL uses a backward arrow, ←, for this purpose, and the same sign is frequently used in assignment statements in flowcharts. Our flowcharts will employ the equals sign. The two statements above appear as they would in FORTRAN or PL/I. In BASIC, we would write the second statement as

```
LET BONUS = 0.05 * SALARY
```
And in COBOL it would appear as

\[ \text{COMPUTE BONUS} = 0.05 \times \text{SALARY} \]

or as

\[ \text{MULTIPLY SALARY BY 0.05 GIVING BONUS} \]

Both statements in example 5.2 are arithmetic assignment statements. The general form of such a statement is

\[ \text{NAME} = \text{arithmetic expression} \]

In response to which computer will

(i) evaluate the arithmetic expression by replacing any variable names by the numerical values stored at those addresses,

(ii) read the value of the arithmetic expression in storage location NAME.

Table 5.1 lists the symbols used in the major languages for the five basic arithmetic operations. Thus, the algebraic expressions:

\[ xy \quad \frac{x}{y} \quad x^y \]

would appear as

\[ X \times Y \quad X / Y \quad X \times Y \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>**</td>
</tr>
<tr>
<td></td>
<td>↑</td>
</tr>
</tbody>
</table>

Table 5.1

Example 5.3

(a) If three test scores are at addresses T1, T2, and T3, the assignment statement

\[ \text{AVERAGE} = (T1+T2+T3)/3 \]

Instructs the computer to add the three scores and divide by 3, and store the result in AVERAGE.

b) Suppose one wants to increase the value of HOURS by 12. This can be done by the assignment statement

\[ \text{HOURS} = \text{HOURS} + 12 \]

This statement may seem strange at the first, since HOURS cannot equal HOURS + 12. However, the equals sign here denotes assignment. The computer first calculates HOURS =12, using the original value in HOURS, and then stores the result back in HOURS. The net effect is that the original number in HOURS is increased by 12.

c) The assignment statement

\[ \text{NEWRATE} = \text{OLDRATE} \]

Instructs the computer to copy the number stored in OLDRATE into the location NEWRATE. In other words, the value of the variable OLDRATE is assigned into the variable NEWRATE.

Input/Output Statements

These statements bring data into the computer from an outside medium or take data from the computer to an outside medium.

Read statement: Suppose that a punched card contains an employee's number, the number of hours that has worked, and his rate of pay. We would write

\[ \text{Read EMPNUMBER, HOURS, RATE} \]
To direct that three number be stored in the computer's memory in locations called EMPNUMBER, HOURS and RATE, respectively.

Write statement: Suppose that memory locations called EMPNUMBER, GROSS and NET contain an employee's number, gross pay, and net pay, respectively. We would write

**Write EMPNUMBER, GROSS, NET**

To direct that the numbers be read out onto some outside medium, such as a pinched card or line printer.

The forms we have given for read and write statements are merely retrospective, the exact depends on the programming language being used.

### 5.3 FLOWCHART SYMBOLOLOGY

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oval</td>
<td>Terminal</td>
</tr>
<tr>
<td>Parallelogram</td>
<td>Input/Output</td>
</tr>
<tr>
<td>Small circle</td>
<td>Connector</td>
</tr>
<tr>
<td>Rectangle</td>
<td>Process</td>
</tr>
<tr>
<td>Diamond</td>
<td>Decision</td>
</tr>
</tbody>
</table>

**Terminal symbol:** This symbol indicates the beginning or end of the program.

**Input/Output symbol:** This symbol usually be filled with either a read or a write statement.

**Connector symbol:** A pair of identically labeled connector symbols denote the indicated points of the flowchart are connected even though no arrow is drawn from one point to the other.

**Process symbol:** This symbol denotes some general processing operation, usually an assignment statement or data movement instruction.

The symbol may also be used to denote a macroinstruction, i.e. a single instruction which normally would require an entire list of commands to implement it.

**Decision symbol:** This diamond shaped box denotes a point in the program where more than one path can be taken. The particular path that is chosen depends on the answer to a question or the result of a test. The question (test) is inserted in the symbol and the arrows leaving the symbol are labeled with the different result, e.g.
with "YES" and "NO" or with "TRUE" and "FALSE" or with "POSITIVE", "NEGATIVE" and "ZERO". One frequently omits the word "Is" within the decision symbol, simply writing:

COUNT ≤ 25?  
Student married?  
SUM = AMOUNT?

Is COUNT <= 25?  
no  
Yes  
Sign of ACCOUNT?  
positive  
negative

Is student married?  
no  
Yes  
Is SUM=AMOUNT?  
no  
Yes

Observe that a decision need not involve numeric data, e.g. "Is student married?". However, if numeric data are involved, then the decision normally involves one of the six mathematical relation given in Table 5.2, or a combination of such relations using one or more of the logical connectives "and", "or", and "not". For example, the question might be:

Is 30 ≤ AGE ≤ 40?

(Which is equivalent to: Is AGE ≥ 30 and AGE ≤ 40?) or

Is A<20 or B <15?

Table 5.2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>≤</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than or equal</td>
</tr>
</tbody>
</table>

Example 5.4  A company plans to give a year-end 3% to each of its employees. However, if an employee has been working 10 or more years at the company, he is to get an additional $50. Figure 5.4 is a flowchart of an algorithm to calculate the BONUS for a given employee. SALARY and YEAR are variables whose values are salary and the number of years with the company.

Fig. 5.4
Example 5.5  The HP company plans to give a year-end 3% bonus to each of its employees earning $10,000 or more per year, and a fixed $300 bonus for the other employees. Figure 5.5 is a flowchart to calculate the BONUS for a given employee. Compare Fig. 5.4.

![Flowchart](image)

The English word "If" appears in two general types of commands:

1) If …….. then
2) If ………then ; else …….

In Example one has the command

\[
\text{If \ YEAR \geq 10 \ then \ BONUS = BONUS +50}
\]

Which is the first type. The flowchart translation uses a decision box, and the statement \( BONUS = BONUS +50 \) is either executed or omitted. On the other hand, in Example one has the statement

\[
\text{If \ SALARY \geq 10,000 \ then \ BONUS = 0.03 \times BONUS \ else \ BONUS = 300}
\]

Which is of the second type. The flowchart translation also uses a decision box, but here only one of the two statements \( BONUS = 0.03 \times BONUS \) \( BONUS = 300 \) is executed. In other words, the first type of command implies that a statement (or block of statements) is executed or omitted, the second type that one of two statements (or blocks of statements) is executed, but not both.

5.4 LOOPS AND THEIR CONTROL BY A COUNTER

Frequently one wants to repeat a process a certain number of times. One way to accomplish this repetition is for the algorithm to transfer control of the computer from the end of the process back to its beginning. The flowchart of such an algorithm will contain a loop; hence the term looping for the repetition of an algorithm or a part of an algorithm.

Example 5.6  Suppose that the Croesus Company (Example 5.1) wants to calculate the bonus for every one of its employees, not just for a given employee. Such an algorithm follows:

Step 1: Obtain (the first or next) employee's name, EMPNAME and his SALARY.
Step 2: \( \text{BONUS} = 0.03 \times \text{SALARY} \)
Step 3: Record EMPNAME and BONUS.
Step 4: Go to step 1.

The flowchart of the algorithm appears in Fig. Observe that the arrows from the Read statement, from the BONUS statement, and from the Write statement form a closed loop.
The flowchart in Fig. 5.6 contains no provision for ceasing to repeat the process (of calculating bonuses). Clearly, one wants to stop when all the employee records have been read and all bonuses have been recorded. One way to "control a loop" i.e. to incorporate a stopping condition, is by using a counter, a variable whose is increased by 1 for each repetition of the process. Figure 5.6 gives a flowchart for an algorithm that is to be repeated 50 times (e.g. the loop of Fig. , if the Croesus Company has 50 employees); the counter is denoted COUNT.

Observe that there are three essential steps in the use of a counter to control a loop:

- **Initialization**: The counter is set equal to one.
- **Test**: Before executing the process, the counter is tested to see if it exceeds its maximum allowable value.
- **Incrementation**: After the process has been executed, the counter is increased by one.

### 5.5 CONTROL OF LOOPS BY HEADER OF TRAILER RECORDS

The algorithm flowcharted in Fig. 8.7 has one main drawback; namely, it will work properly only if there are exactly 50 records to be processed. Clearly, the number of records in a file may vary from time to time, and one would like an algorithm which is independent of that number. Such an algorithm is possible if the file provided with a header or trailer record.
Control by a Header Record
Assuming that the header record contains (among other things) the number \( N \) of other records in the file, one can first read this number, and then use \( N \) as the end value of a counter controlling a loop that processes the file. Figure 5.8 gives such an algorithm.

Control by a Trailer Record
The trailer record is assumed to contain (among other things) a signal it is the last record in the file. Thus, on reading each record, we may ask: "Is this the end of the file?". An algorithm based on this test is depicted in Fig. 5.9.

![Flowchart of Control by a Trailer Record](image)

Example 5.7 A common way of signaling EOF is to have all 9s in some data field of the trailer record. Figure 5.10 gives an algorithm for calculating and recording the bonus of each employee in Example 5.1.

![Flowchart of Example 5.7](image)

We note that the use of a header record to control a loop requires the number \( N \) records being processed, a number that may change very often (e.g. if patients are continually being admitted or discharged from hospital). In such a case, it might be more advantageous to use a trailer record to control the loop, rather than recalculate the end value \( N \) each time the file is updated.

5.6 ACCUMULATORS
Frequently one wants to find the sum of certain numerical quantities that are read or calculated one at a time. This can be accomplished by first choosing a variable, say \( \text{SUM} \), and initializing it to zero:

\[
\text{SUM} = 0
\]

One then executes the instruction

\[
\text{SUM} = \text{SUM} + \text{QUAN}
\]
As each numerical quantity, called QUAN, is obtained. The variable SUM is called an accumulator because it accumulates all the numerical quantities. The final value of SUM is then the required value.

**Example 5.8** Figure 5.11 is the flowchart of an algorithm for summing the first twenty integers. There we use SUM as the accumulator, and the variable K to produce the numbers 1, 2, 3, …, 20.

![Flowchart](image)

Observe that

\[
\text{SUM} = 0 \quad \text{first assigns} \quad 0 \text{ to SUM.}
\]

When K = 1,  \( \text{SUM} = \text{SUM} + K \) assigns \( 0 + 1 = 1 \) to SUM

When K = 2,  \( \text{SUM} = \text{SUM} + K \) assigns \( 1 + 2 = 3 \) to SUM

When K = 3,  \( \text{SUM} = \text{SUM} + K \) assigns \( 3 + 3 = 6 \) to SUM, and so on.

When K = 20, the value of SUM is 190, so

\[
\text{SUM} = \text{SUM} + K \quad \text{assigns} \quad 190 + 20 = 210 \text{ to SUM}.
\]

This is the value of SUM when the Write statement is executed, so the output of the program is 210, which is the sum of the integers from 1 to 20.

**Example 5.9** Suppose that a company wants to find the total salary paid to its employees of a company, Figure 5.12 is the flowchart of an appropriate algorithm.

Observe that the accumulator SUMSAL is initialized before any record is read, and its value is printed only after the records have been read. The flowchart also assumes that the file has a trailer record in which -999999.99 is punched in the salary field.

![Flowchart](image)
5.7 **TWO SPECIAL ALGORITHMS:**

Given a list of numerical values, say the salaries of the employees of a company, we wish to find:

a) The average of the salaries,

b) The largest salary.

We will assume that the file of the employee records contains a trailer record, so that we may use an EOF decision box in our flowcharts.

**Average Value**

The idea behind this algorithm is first to find the sum of the salaries, using an accumulator, say SUM. Because the number N of employees is not assumed to be known in advance, the algorithm uses a counter to count the records read. The average value, say AVG, is then SUM divided by N. Figure 5.13 is the flowchart of our algorithm.

**Largest Value**

In this algorithm we first set a variable, say BIGSAL, equal to the first salary. Then we compare BIGSAL to each successive salary. If BIGSAL exceeds or equals a salary, then we do not change BIGSAL; however, if BIGSAL is less than a salary, then we assign this larger salary to BIGSAL. The final value of BIGSAL is the largest salary. Figure 5.14 is the flowchart algorithm.
5.8 DO LOOPS

It is convenient to introduce a special type of loop that is headed by a special macroinstructions. We write the statement

\[ \text{Do } K = 1 \text{ to } N \]

Before a list of instructions, called the body of the loop, as in Fig. 5.15. We call the resultant loop a DO loop. This terminology comes from FORTRAN, although many programming languages have this type of loop.

For example:

\begin{align*}
\text{BASIC} & \quad \text{DO } K = 1 \text{ to } N \\
& \quad \{ \text{body of loop} \} \\
& \quad \text{END;}
\end{align*}

\begin{align*}
\text{FORTRAN} & \quad \text{Do } n \ K = 1 \ , \ N \\
& \quad \{ \text{body of loop} \} \\
& \quad n \ \text{CONTINUE}
\end{align*}

The precise meaning of the above loops is given by the flowchart in Fig. 5.16(a) or in Fig.5.16(b), depending on which language and which compiler is used. In brief, the loops instruct the computer to execute the body of the loop first with \( K = 1 \), then with \( K = 2, \ldots \), and finally with \( K = N \). The variable \( K \) is called the index, the number 1 the initial value, and \( N \) the end value of the loop. In a more general type of DO loop, the initial value need to be 1 and the index need not increase in step of 1.

The index itself can be one of the variables in the body of a DO loop. This fact is especially important when one or more of the other variables are arrays.

**Example 5.10**  
A company has 80 employees. The flowchart in Fig. 5.17 gives an algorithm that finds the average salary and the number of employees earning above the average salary. Observe that the salaries are read into an array, \( \text{SALARY} \). Next, the average salary, \( \text{AVG} \), is calculated. Then each salary, \( \text{SALARY}(K) \), is compared with \( \text{AVG} \) to obtain the number \( \text{NUM} \) of salaries greater than \( \text{AVG} \).
5.9 SOLVED PROBLEMS

5.1 Consider the following flowchart in Fig. 5.18. Find the output if the input is

a) A = 5, B = 10
b) A = 10, B =
   a) Since A less than B, BONUS is not increased in value, and 300 is outputted.
   b) Since A is greater than B, BONUS = BONUS + 100 is executed, which increases the value of BONUS by 100. Hence, 400 is outputted.

5.2 Consider the following flowchart in Fig. 5.19. Find the output if the input is

a) A = 5, B = 10, C = 15
b) A = 15, B = 5, C = 10
   a) Since A is greater than B, but B is less than C; so BONUS = 200 is executed, and 200 is outputted.
   b) A is greater than B and B is greater than C; so BONUS = 300 is executed, and 300 is outputted.
5.3 The flowchart in Fig. 5.20 uses connector symbols labeled A, B, or C. Two or more of the symbols can have the same label, but only one of the like-labeled symbols can have an arrow exiting. Find the output if the input is:

a) \( X=15 \), \( Y=20 \)

b) \( X=5 \), \( Y=5 \)

c) \( X=1 \), \( Y=4 \)

d) \( X=15 \), \( Y=2 \)

a) The answer is "no" to "\( X<10 \)" and to "\( Y<10 \)"; hence \( Z=X+Y \) is executed, and \( 15+20 = 35 \) is the output.

b) The answer is "yes" to "\( X<10 \)" so control is transferred to "\( X=X+5 \)" by way of connector symbol labeled A. This increases the value of \( X \) to 10; hence the answer is "no" to "\( X<8 \)". Thus \( Z = 5\times X = 5\times 10 = 50 \) is executed. Control is transferred to "Write Z" by way of connector symbol labeled C, so that 50 is the output.

c) The answer is "yes" to "\( X<10 \)"; hence "\( X=X+5 \)" and the value of "\( X \)" is increased to 6. The answer is "yes" to "\( X<8 \)"; hence "\( Y=Y+5 \)" is executed, and the value of \( Y \) is increased to 9. The answer is "no" to "\( Y<8 \)"; so \( Z = 5\times Y = 5\times 9 = 45 \) is executed and 45 is the output.

d) The answer is "no" to "\( X<10 \)"; but is "yes" to "\( Y<10 \)"; hence "\( Y=Y+5 \)" is executed, and the value of \( Y \) is increased to 7. The answer is "yes" to "\( y<8 \)"; so control is transferred back to "\( Y=Y+5 \)" is executed, and the value of \( Y \) is increased to 12. Now the answer is "no" to "\( Y<8 \)"; so \( Z = 5\times Y = 5\times 12 = 60 \) is executed, and 60 is the output.

5.4 Draw flowcharts that interpret the following statements:

a) If \( \text{ACCOUNT} \) is 10,000 or more, then add 500 to \( \text{ACCOUNT} \)

b) If \( \text{ACCOUNT} \) is 10,000 or more, then add 500 to \( \text{ACCOUNT} \); else add 300.

![Flowchart](image)
5.5 Consider the equation \( Y = X^3 - 8X^2 + 5X - 4 \) Draw a flowchart which calculates \( Y \) for values \( X \) from \(-4\) to \(4\) step \(0.2\), and prints each \( Z \) and its corresponding \( Y \).

![Flowchart](image)

5.6 Find the output for each flowchart in Fig 5.23.

![Flowchart](image)

5.7 Let \( WAGE \) be a linear array giving the salaries of \( N \) employees. Give an algorithm that finds the largest salary and who earns it, i.e. that finds the subscript \( L \), assumed unique, such that \( WAGE(L) \) is largest.

First set \( L=1 \) and set \( \text{Max}=WAGE(1) \). Then compare \( \text{Max} \) to each \( WAGE(K) \). If \( \text{Max} \) is less than \( WAGE(K) \) reset \( L=K \) and reset \( \text{Max}=WAGE(K) \). The flowchart of algorithm appears in Fig. 5.24.
5.10 REVIEW QUESTIONS

5.1 Draw a flowchart of an algorithm that calculates the area of circumference of the circle.

5.2 Draw the flowchart to determine the values of $A$, $B$, and $C$ in the following questions:

\[ A = x^2 + 2y \]
\[ B = 2x - 3A \]
\[ C = A^2 + xB \]

5.3 Draw the flowchart to find the sum of the first 50 natural numbers.

5.4 Draw the flowchart of the algorithm which finds a general solution for the question:

\[ ax^2 + bx + c = 0 \]

5.5 A company's salesmen are selling toothpaste and tooth powder. The company having 50 salesmen gives 10% commission on the sale of the toothpaste and 20% commission on tooth powder. Draw the flowchart to compute and print the total sale and the total commission for each salesman.

Note: $S$: salesman number

- $P_1$: amount of sales of paste
- $P_2$: amount of sales powder
- $K$: counter for the number of salesman

5.6 How many times, and for which values of the index, will the DO loops whose headings follow be cycled?

a) Do $K=3$ to 24 by 5
b) Do $K=6$ to 17
c) Do $L=8$ to 4 by 3
d) Do $M=2$ to 200 by 3
5.7 Consider the equation \( y = x^4 + 2x^3 - 13x^2 - 14x + 24 \). Draw a flowchart that calculates \( y \) for values of \( x \) from -5 to 5 in step of 0.25, and writes each \( x \) and its corresponding \( y \).

5.8 Draw a flowchart for computing factorial N (N!)

5.9 Suppose the monthly telephone bill is to be computed as follows:

- **Minimum** $7.50 for first 80 message units
- **Plus** 0.06/unit for the next 60 units
- **Plus** 0.05/unit for the next 60 units
- **Plus** 0.04/unit for any units beyond 200

Draw a flowchart which calculates the monthly BILL, with input MESSAGE, the number of message units.