Compiler	Interpreter
Compiler translates whole program at a time.	Interpreter translates and executes program line by line.
The syntax errors are known before program is executed as program is translated only if there are no errors.	The syntax errors are known only when the statement containing error is translated. It terminates once the error occurs. Hence, further errors are known only during next interpretation.
The first output comes only after successful translation of whole program and the execution.	The first out comes immediately as translation and execution is simultaneous.
After successful translation, for every execution no translation is required.	Translation is done for every execution.
It is faster as only translation is done once.	It is slower as translation is done every time.
Language like C, C++ uses compiler.	Language like BASIC, Java uses interpreter.

2.2 Characteristics of Higher Level Languages :

The major common characteristics of higher level programming languages are listed below :

- Interactive
- Variety of data types
- Rich set of operators
- Flexible control structures
- Readability
- Modularity
- File handling
- Memory management
- Procedural Vs. object-oriented
- Procedure or event-driven
- Interface to other languages
- Availability of library support
- Supporting tools

These are discussed briefly here to give idea of features found in higher level languages. Sometimes, C is taken as example to illustrate things more precisely. The list is any way not an exhaustive.

comma, dot, arrow, sizeof and others.

Flexible control structures :

The control structures are providing necessary mechanism for implementing algorithms in language format. The structured programming languages provide mainly three types of control which are essential for any types of programming. They are

- **sequence**
- **selection or decision**
- **repetition or looping**

Any algorithm either simple or complex consists of various combinations of above three types of steps. Thus, in order to implement such algorithms, language should provide necessary control structures which implement above.

The sequence is inherent to any programming language as control always goes to next statement in sequence unless otherwise, it is intentionally send to some other part. For example, following C statements illustrate sequence.

```
x = 5;
y = x+10;
```

In above case, first statement assigns 5 to **x** and then control moves sequentially to next statement which assigns value of expression x+10(=15) to **y**.

The selection or decision statements are very important and used for deciding further execution path based on logical conditions. A decision may skip some part of program. For example, two-way decision selects one of the block or group of statements out of two. Multi-way decision selects one from many. The C programming language supports number of selection and decision statements like **if**, **if-else**, **multiple if-else**, **switch** etc.. A simple example of two way decision is

```
if(x > y)
    printf("%d",x);
else
    printf("%d",y);
```

In above case, depending on values of **x** and **y**, only one value which is larger will be printed. Hence, decision is made by comparing values of **x** and **y**.

Many times, it is necessary that same process is to be repeated. For this purpose, languages support three different looping statement as follows :

- **for loop**
- **while loop**
- **do-while loop**

Following example, repeats process of printing value of variable **i** for ten times.

```
for(i=0;i<10;i++)
    printf("%d\n",i);
```

The modern and very popular languages like Java, Python started supporting features that allow very compact codes, thus saving developers time and making development faster.

2.4.1 Algorithms :

An algorithm is a stepwise solution to a problem. Each step in an algorithm represents a solution to a small problem. Hence, algorithm gives a sequence of steps which makes complete solution of a problem in hand. An algorithm itself is decomposition of problem into small steps which are easily understandable. Consider the following examples.

EXAMPLE 1

Problem : *Write an algorithm to find area of circle.*

In this problem, first radius of circle is known. Once radius is known, it can be used to compute area as

```
A = PIE * R*I
```

which finally is given as output. An algorithm is given below.

Algorithm :

1. Input R
2. Compute A = 3.14 * R * R
3. Print A
4. STOP

Each step starts with unique number (which are in increasing order) followed by operation. Operation uses appropriate word and quantities (constants or variables). For example,

```
1. Input R
```

has **1** as step number and **Input R** represents that a value of **R**(radius) is to taken from user. The last step is **STOP** which denotes end of algorithm. The words i.e. Input, Compute, Print should denote appropriate action and user is free to use any meaningfull words. Example 1 is purely sequential problem.

EXAMPLE 2

Problem : *Write an algorithm to find minimum of two numbers.*

This requires two numbers N_1 and N_2 to be read, compared and whichever is smaller is to be printed. Algorithm is as follows :

Algorithm :

1. Input N_1, N_2
2. If $N_1 < N_2$, goto 5
3. Print N_2
4. goto 6
5. Print N_1
6. STOP

Step 1 reads two numbers N_1 and N_2. In step 2, N_1 and N_2 are compared. If N_1 is smaller than N_2, condition $N_1 < N_2$ is true and next step done is step 5 where N_1 is printed. If $N_1 < N_2$ is false, then next step done is step 3 which prints N_2. After printing N_1 or N_2, it is ended with step 6.

Algorithms are written in physical sequence, but they are evaluated in logical sequence. Step 1 to 6 in increasing order defines a physical sequence in example 2. Logical sequence which is used to evaluate, depends on the input values given. There may be more than one logical sequence where each corresponds to exactly one possible input combination. Example 2 is evaluated as follows :

Case-1 :

1. $N_1 = 5$, $N_2 = 7$
2. $N_1 < N_2$, goto 5
5. Print 5
6. STOP

The sequence of steps used here are 1, 2, 5 and 6. It forms a logical sequence or a path through which evaluation is made. This path is taken as input value $N_1 = 5$ is less than $N_2 = 7$.

Case-2 :

1. $N_1 = 7, N_2 = 4$
2. $N_1 < N_2$ (next step, as it is false)
3. Print 4
4. goto 6
6. STOP

as $N_1=7$ which is greater than $N_2 = 4$. Logical sequence followed is 1, 2, ,3, 4 and 6. For, $N_1 = N_2$ also, the same sequence is followed.

Hence, algorithm of example 2 has two logical sequences inside it :

1. 1, 2, 5, 6
2. 1, 2, 3, 4, 6

This defines set of possible paths for evaluation. As size and complexity of algorithm increases, number of paths increases. But remember that for given input, there will be always a one path. Algorithm once translated to higher level language statements, it is called a **program**. A program is defined as a logical sequence of instructions. Instructions are done in logical order as discussed for algorithms. A point or step at which currently evaluation (or execution) is being performed is called **control**. Control always flows in logical sequence. Normally it goes in sequential order, but decision making, looping or flow breaking statements causes it to alter the normal sequence as per requirement.

structures sequence, decision making and repeatition (or looping) respectively.

2.4.2 Flowcharts :

A flowchart is a pictorial representation of an algorithm. As it represents solution in form of picture, it is more easier to understand and develop. A main advantage of flowchart is visibility of paths (or flows) within solution. Each path (or logical sequence) is clearly visible as arrows are used to represent flow. Before developing flowcharts for example problems, different flowchart symbols must be known. They are shown in fig 2.3.

Last two symbols in fig. 2.3 are useful when flowchart does not fit into single page.

For example, symbol

indicates break in current page and corresponding symbol in other page

denotes continuation of path from another page. Symbol "A" inside circle should match, if they both represents same path.

Following are three examples of flowcharts which are developed for algorithm written in 2.4.1.

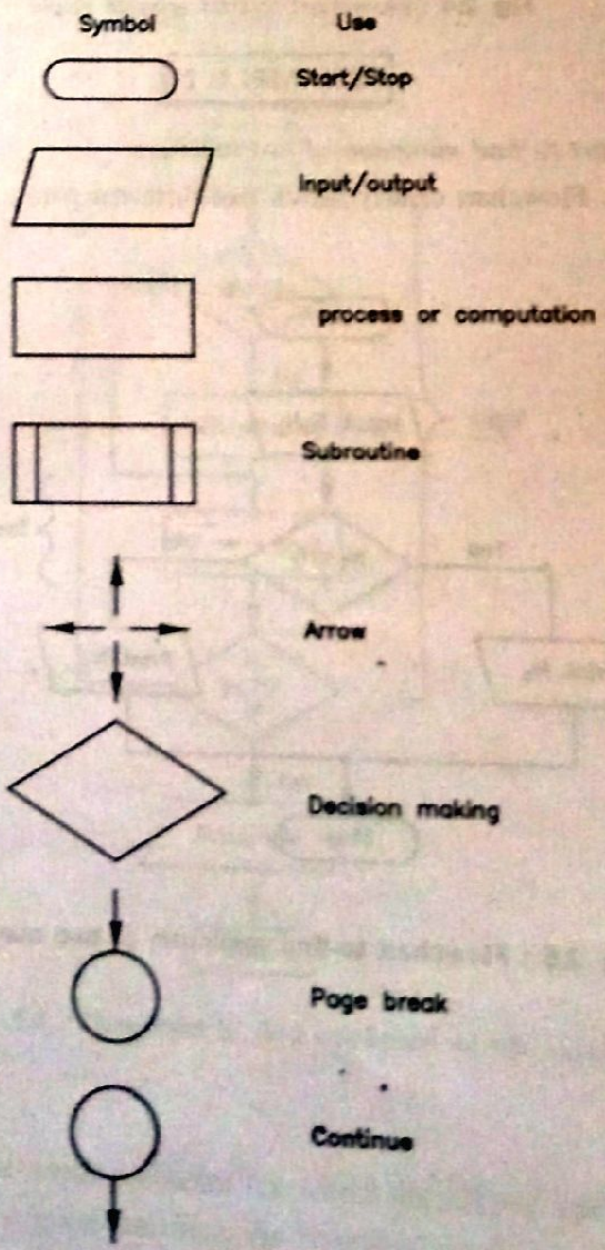


Fig. 2.3 : Flowchart symbols

Problem 1 : *Find the roots of algebraic equation using*

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Algorithm :

1. Input a, b, c
2. Compute t1 = b * b – 4 * a * c
3. Compute t2 = SQRT(t1)
4. Compute t3 = 2 * a
5. Compute x1 = (– b – t2) / t3
6. Compute x2 = (– b + t2) / t3
7. Print x1, x2
8. STOP

Fig 2.7 shows the flowchart.

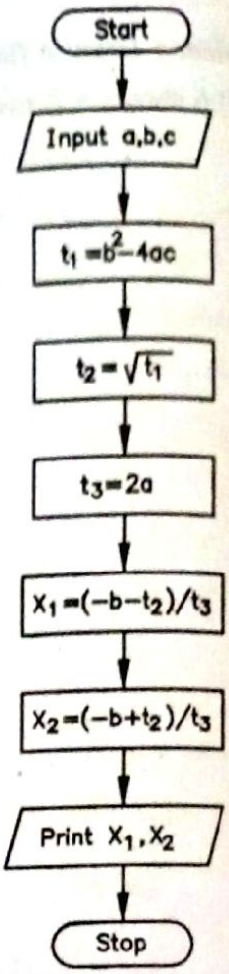


Fig. 2.7 : Flowchart to find roots

It is also possible to combine several steps into single step to reduce the size of algorithm and/or flowch Fig 2.8 shows alternative flowchart for same problem.

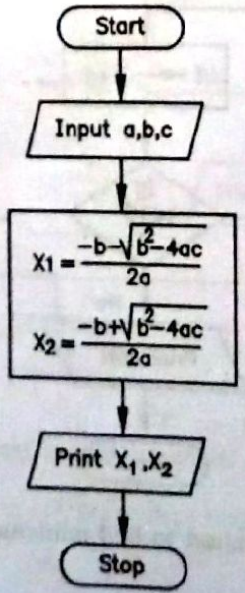


Fig. 2.8 : Flowchart with less steps for problem 1

read this page only one time

3.0 Introduction :

The count of persons desiring to learn programming increased in last few years as awareness about computer has gone high with easy access of computers not in offices but in homes also. It is fact that today most of the people who want to learn the computer programming start with C programming. The major reason behind it is the wide spread use of C programming since last 30+ years. The C is mainly used in system programming as the program or software written in C is compact and faster. Additionally, it is portable which makes use of program written in C on any hardware possible i.e. from PC(Personal Computer) to supercomputer.

3.1 History of C :

The C is widely used programming language. The C and Unix are associated with each other right from beginning. The Unix which was written in C is very popular operating system. Around 1969, the Unix was developed by the Ken Thompson and Denis Richie in Bell Laboratory, America for the PDP series of machines. The strong features and performance of this system attracted the other people in laboratory to work on it, but it was not possible as it was written in machine language of the PDP machines.

To solve this portability problem, Denis Richie decided to develop such a language which is portable means the program written in that language runs on any machine having different hardware. At that time the language B was popular which was derived from the language BCPL. Denis Richie developed the today's most popular language C from the language B.

Once C was completely developed, they rewrote the Unix in C. That made it possible to port Unix on any hardware as C is portable. Today Unix is available on PC to supercomputers. The C is always part of Unix on all Unix systems. The C is also available on almost all the operating systems.

3.2 Features of C :

The main features of C language are as follows.

- C is portable.
- C does not contain Input/Output statements.
- C supports Bit-wise operators which normally are not supported by higher level languages.
- C is modular language.

The C was developed to solve the portability problem of Unix. Hence, program written in C is always portable. Portable means a C program written on one hardware works on other hardware without modifications or with minor hardware dependent modifications. The portability is very strong feature of C programming.

Normally program written in higher level languages are interactive. For that purpose, programming languages support I/O statements. For example, BASIC language contains INPUT statement to read the input from the keyboard and PRINT statement to produce the output on screen. But C does not support such I/O statements. For this purpose C uses the library functions like scanf() and printf().

The C also supports the bit-wise operators to perform the bit-wise operations like AND, OR, NOT etc. Normally, these operations are not supported by the higher level languages, but are common in machine/assembly languages. Because of these reasons (no I/O statements and support of bit-wise operations), C is not considered as fully higher level language, but known as middle level language.

The programs in C are written in form of functions. A C program is group of one or more functions. Dividing program into small functions, makes it easy to develop and maintain the programs. Each function is easy to understand as it is small. The method of dividing a program into small functions is called *modularity*. The modularity is essential part of the structured programming. Hence, C is also example of structured programming.

3.3 Structure of C Program :

Before we go into details of C programming, let us understand structure of a C program by writing small C programs.

3.4 C Character Set and Tokens :

The words(tokens) and statements used in any language are formed from their basic character set. For example, the words used in English language are formed from its alphabet having 26 different symbols i.e. A,B, ... , Z or a,b, ... , z. The characters used in C are divided into four categories.

- **Letters or alphabets**
- **Digits**
- **Special characters**
- **White spaces**

The letters include the uppercase(A,B, ... ,Z) and lowercase(a,b, ... , z) alphabets of English language. The digits include 0,1,2, ... , 9. The C also uses special characters like ;(semicolon), '(single quotes), "(double quotes), +,–,*,/,%,>, = etc. for different purposes. The white spaces are used to separate the words or tokens. They are blank, tab and newline.

The character together makes special symbol or word knows as *token*. The examples of tokens are the words used to define data types like **int**, **float**, etc. Similarly any operator i.e. +,–,*,/,%,> etc, punctuation marks like ;(semicolon), braces { } etc are also tokens. The constants used in programs like 12, 12.65 are also tokens.

3.5 Keywords :

The keywords are the special words used in C programming having specific meaning. The meaning of these words can not be changed. They are also known as the reserve words. Table 3.1 gives the list of C keywords as per the ANSI standard. These are reserve words and can not be used as identifiers.

Table 3.1 : keywords

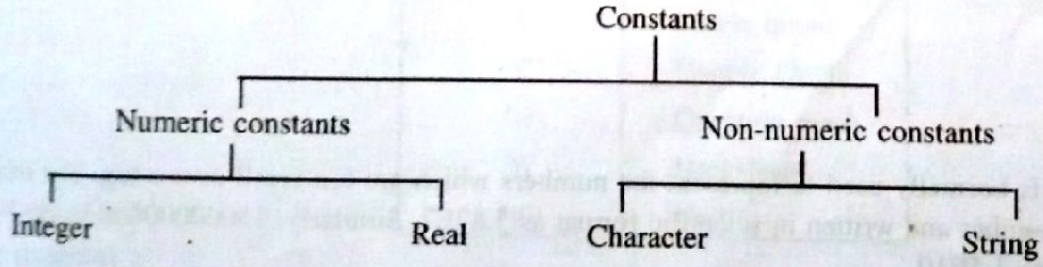
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

3.6 Identifiers :

The *identifiers* are user defined names used in programs for providing names to variables, arrays and functions. The identifiers are made up of letters(uppercase and lowercase), digits and underscore. The C is case sensitive and hence the uppercase and lowercase letters are not treated as same. For example, **max** and **Max**, are not same. Normally, the lowercase letters are used in identifiers, but is not the limitation as uppercase can also be used. The keyword listed in table 3.1 has special meaning to C and can not be used as identifiers.

3.7 Constants :

The *constants* are the values which never changes. The C uses following types of constants.



Integer constants :

Integer constants represent the whole numbers. They uses digits 0 to 9 and optional sign + or – before the number. The following are examples of valid integer constants.

123
–23
65535
0
+85

The space, comma and other special symbols are not allowed. The following are examples of invalid integer constants.

10,000
$500
15 780

The C also allows the octal and hexadecimal integer constants in addition to decimal constants. The octal numbers are written with preceding 0 and hexadecimal numbers are preceded by 0x or 0X. The following are examples of valid octal and valid hexadecimal constants.

Octal	Hex
0177	0x32ab
05	0X92F
0831	0x1234

For unsigned integer constants, number is followed by 'u' or 'U'. For long integer constants, number is followed by 'l' or 'L'. Consider the following examples.

1234u	or	1234U	(unsigned integer)
123456l	or	123456L	(long integer)
5432178ul	or	5432178UL	(unsigned long integer)

Real constants :

Real constants represent the numbers with fractional parts i.e. digits after decimal point. They are used to represent contiguous quantities like temperature of day. The following are examples of valid real constants.

0.0005
–0.123
+248.0
215.
.78

Real constants can be written in scientific formats also. For example, 314.28 is written in scientific format as 3.1428e2. The following are examples of real constants in scientific format.

6.8e+4
–8.92E–1
1.7e+2
8.19E3

The scientific format is normally used to represent the numbers which are too small or too big. For example, 0.000000587 is very small number and written in scientific format as 5.87E7. Similarly, 15000000000 is too big and written in scientific format as 1.5E10.

Character constants :

Character constants represent single character and are always enclosed in single quotes. The following are examples of valid character constants.

'A' 'a' '?' ':' '8'

The characters are stored in computer memory in form of their ASCII values. For example, ASCII value of 'A' is 65 and ASCII value of '8' is 56. Study the program 3.4 to clarify the concept.

3.8 Variables :

The *variables* represent the quantities which changes with the time. Each variable identified by a unique identifier in a program called variable name. The variables are used to store values which changes continuously. The **length, breadth, area** etc. are examples of variables which we have used in earlier programs.

The ANSI rules for variables names are as follows.

1. Only uppercase and lowercase letters, digits and underscore can be used.
2. Variable name always begins with letter.
3. Only first 32 characters are significant.
4. Keywords can not be used.
5. Variable names are case sensitive. The **Count** and **count** are two different variables.

The care should be taken while choosing the names of variables, so that the names are appropriate to the quantities to be represented and meaningful. Consider following variable names.

average

count

maximum

temp_calc

temp_fahr

✓ The underscore character is used to separate the words in a variable name when name consists of multiple words. The digits are useful to have sequence of names with each name has same meaning, but digit differentiates the instances. Table 3.3 shows the some valid or invalid examples.

Table 3.3 : Examples of variable names

Variable name	Valid/Invalid	Remark
max	Valid	
max4	Valid	
max 3	Invalid	Space is not allowed
avg_num	Valid	
double	Invalid	Reserve word
1temp	Invalid	Should start with letter

3.9 Data Types :

There are four basic data types supported by C. Table 3.4 lists the basic data types with their size in bytes and the respective range of values.

Table 3.4 : Basic data types

Type	Size in bytes	Range
char	1	-128 to +127
int	2	-32768 to +32767
float	4	3.4e-38 to 3.4e+38
double	8	1.7e-308 to 1.7e+308

The C provides different qualifiers like **signed, unsigned, short and long**. By applying the qualifiers to the basic data types, we can change the range of our data type as per the need. Table 3.5 lists the various data types available by applying the qualifiers to the basic data types with their size and range of values.

Table 4.7 : Precedence and associativity

Operators	Associativity
( ) [] -> .	left to right
! ~ ++ -- + - * & (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	left to right
= += -= *= /= %= ^= |= <<= >>=	right to left
,	left to right

PROGRAM 5.2

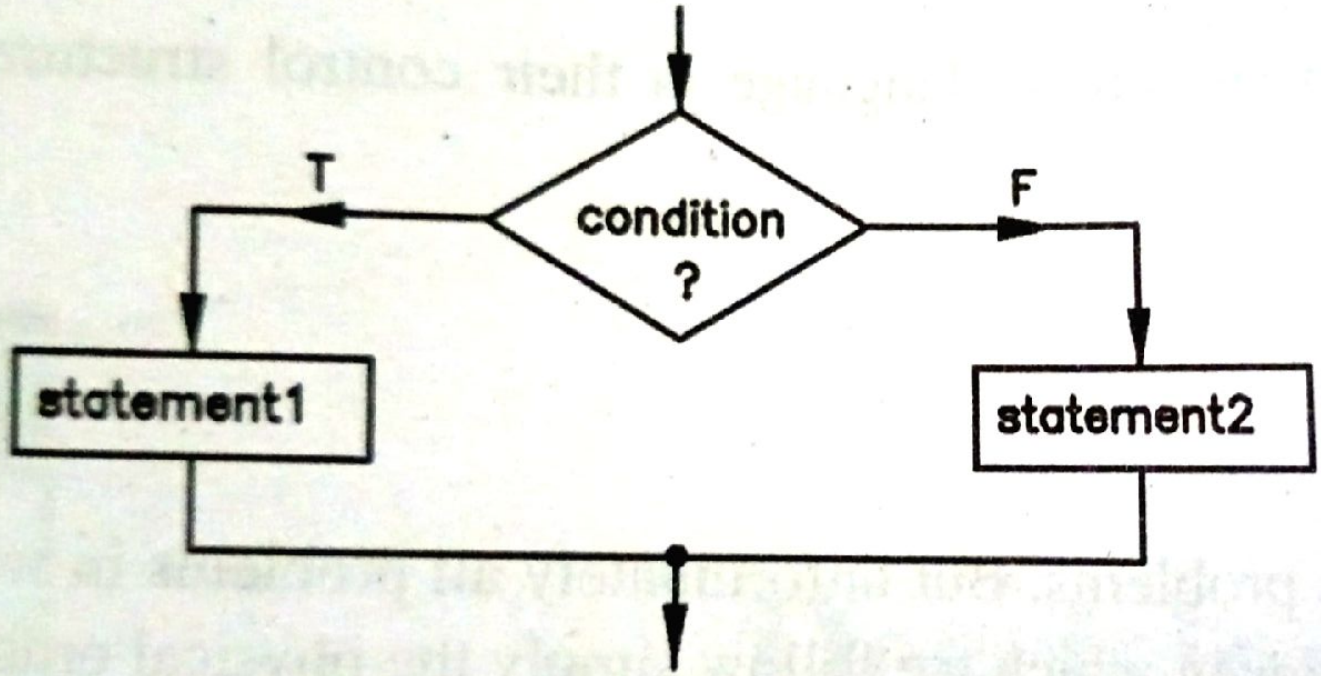
Write a program to compute the weight average(ω), when pessimistic(α), most likely(β) and optimistic(γ) values are given. The weighted average is computed as

$$\omega = \frac{(\alpha + 4\beta + \gamma)}{6}$$

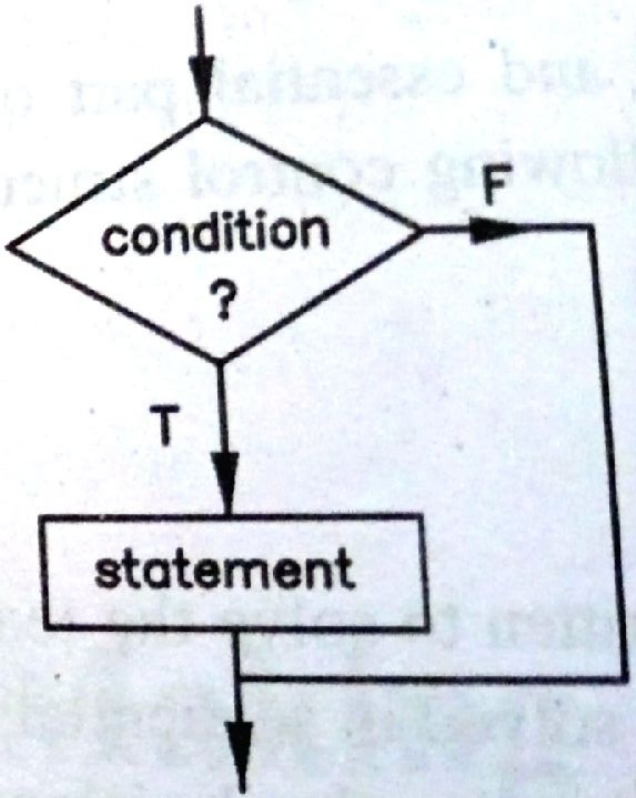
```
#include <stdio.h>

void main()
{
    double alpha,beta,gamma,weight_avg;

    printf("Enter the pesimistic value : ");
    scanf("%lf",&alpha);

    printf("Enter the most likely value : ");
    scanf("%lf",&beta);

    printf("Enter the optimistic value : ");
    scanf("%lf",&gamma);

    weight_avg = (alpha+4*beta+gamma)/6;

    printf("Weight Average = %lf\n",weight_avg);
}
```

RESULT

```
Enter the pesimistic value : 4.5
Enter the most likely value : 6.7
Enter the optimistic value : 7.8
Weight Average = 6.516667
```

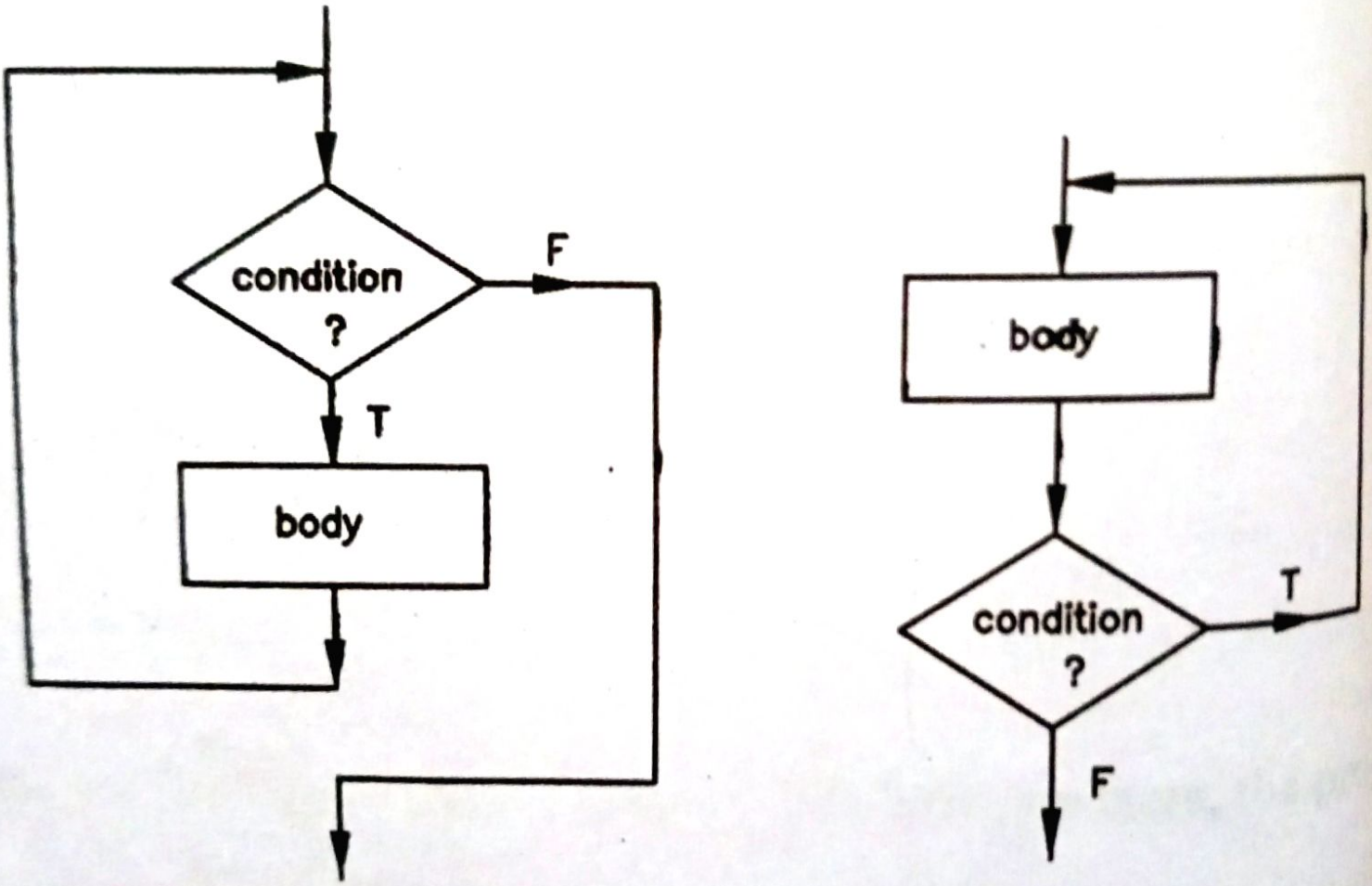


(a)



(b)

Fig. 6.2 : (a) Flowchart of if-else (b) Flowchart of if



(a) Entry control

(b) Exit control

Fig. 7.1 : Flowchart of Loops