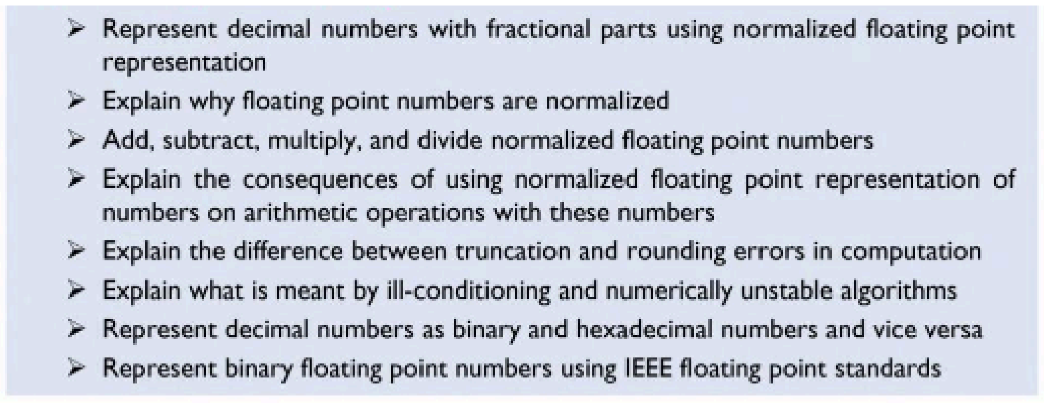
IT-504: Computer Arithmetic: Learning Outcome

(referenceL V. Rajaraman, PHI Publications)

Expected Outcome



- ➢ Represent decimal numbers with fractional parts using normalized floating point representation
- ➢ Explain why floating point numbers are normalized
- ➢ Add, subtract, multiply, and divide normalized floating point numbers
- ➢ Explain the consequences of using normalized floating point representation of numbers on arithmetic operations with these numbers
- ➢ Explain the difference between truncation and rounding errors in computation
- ➢ Explain what is meant by ill-conditioning and numerically unstable algorithms
- ➢ Represent decimal numbers as binary and hexadecimal numbers and vice versa
- ➢ Represent binary floating point numbers using IEEE floating point standards

Introduction to Numerical Algorithms

The main features of a computer which influence the the formulation of algorithms are:

1. The algorithm is stored in the memory of the computer.This facilitates the repetitive execution of instructions.
2. Results of computation may be stored in the memory and retrieved when necessary for further computation
3. The sequence of execution of instructions may be altered based on the results of computation.The facility to rest the sign of a number or test if it is zero coupled with rhe presence of the entire algorithm computer’s memory enables alternate routes to be taken during the execution of the algorithm.
4. The computer has the capability to perform only the basic arithmetic operations of addition,subtraction,multiplication and division.

In Numerical Analysis together with representation of numbers (Integers and Real), there are 2-types of arithmetic operations that are available in a computer.

1. Integer Arithmetic
2. Real /Floating point Arithmetic

Integer Arithmetic

Integers are signed numbers without fractional parts.
Integers are mainly used in counting and subscripts
Integer Arithmetic deals with integer operands

Integer Arithmetic

Real arithmetic uses numbers with fractional parts as operands.
They are used in most computations.

Floating Point representation of numbers

There are two types of arithmetic operations available in a computer. These are:

(i) Integer arithmetic
(ii) Real or floating-point arithmetic.

Integer arithmetic, as the name implies, deals with integer operands, that is, numbers without fractional parts. It is used mainly in counting and as subscripts.

Real arithmetic uses numbers with fractional parts as operands and is used in most computations.

Due to economic considerations computers are designed such that each location (also called a *word*) in memory stores only a finite number of digits. Consequently all operands in arithmetic operations have only a finite number of digits. For illustrative purpose we will assume that a (hypothetical) computer has a memory in which each location can store 6 digits and has provision to store one or more signs. One method of representing real numbers in such a computer would be to assume a fixed position for the decimal point and store all numbers (after appropriate shifting if necessary) with an assumed decimal point as shown in Figure 2.1.

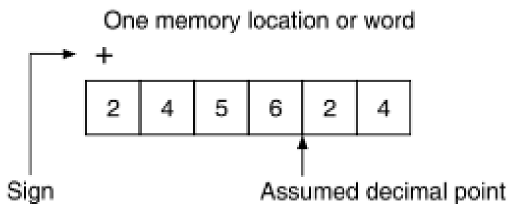


Figure 2.1 A memory location storing the number 2456.24.

If such a convention is used the maximum and minimum (in magnitude) numbers that may be stored are 9999.99 and 0000.01 respectively. This range is quite inadequate in practice and a different convention for representing real numbers is adopted. This convention aims to preserve the maximum number of significant digits in a real number and also increase the range of values of real numbers stored. This representation is called the *normalized floating point* mode of representing and storing real numbers. In this mode a real number is expressed as a combination of a *mantissa* and an *exponent.* The mantissa is made less than 1 and greater than or equal to 0.1 and the exponent is the power of 10 which multiplies the mantissa. For example the number 44.85×10^6 is represented in this notation as 0.4485E8 (E8 is used to represent 10^8). The mantissa is 0.4485 and the exponent is 8. The number is stored in a memory location as shown in Figure 2.2.

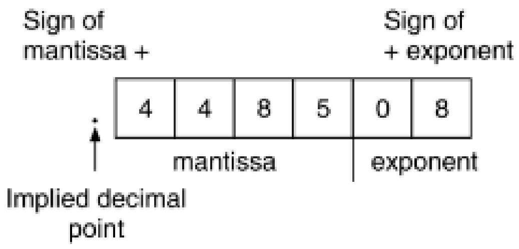


Figure 2.2 Representation of 0.4485E8 in normalized floating-point mode.

In the above case the 6 digits available in a memory location are arbitrarily divided into two parts. Four digits are used for the mantissa and two for the exponent. The mantissa and the exponent have their own independent signs. While storing numbers the leading digit in the mantissa is always made non-zero by appropriately shifting it and adjusting the value of the exponent. Thus the number 0.004854 would be stored as shown in Figure 2.3. The shifting of the mantissa to the left till its most significant digit is non-zero is called *normalization*. The normalization is done to preserve the maximum number of useful (information carrying) digits. The leading zeros in 0.004854 serve only to locate the decimal point. This information may thus be transferred to the exponent part of the number and the number stored as 0.4854 E–2.

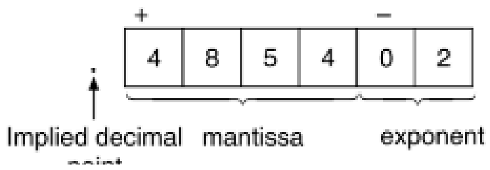


Figure 2.3 Representation of 0.004854 in normalized floating-point mode.

When numbers are stored using this notation the range of numbers (magnitudes) that may be stored are 0.9999×10^{99} to 0.1000×10^{-99} which is obviously much larger than that used in the fixed decimal point notation described earlier. This increase in range has been obtained by reducing the number of significant digits in a number by 2.

Floating Point Arithmetic

Addition

2.3.1 Addition

If two numbers represented in normalized floating-point notation are to be added the exponents of the two numbers must be made equal and the mantissa shifted appropriately. The details will be clarified by some examples.

EXAMPLE 2.1

Add 0.4546E5 to 0.5433E5. In this case the exponents are equal. Thus the mantissas are added. The sum is thus 0.9979E5.

EXAMPLE 2.2

Add 0.4546E5 to 0.5433E7. In this case the two exponents are not equal. The operand with the larger exponent is kept as it is and the mantissa of the operand with the smaller exponent is shifted *right* by a number of places equal to the difference in the two exponents. In this example the difference between the exponents is (7 – 5) = 2. Thus the mantissa of 0.4546E5 is shifted right two places. Each shift causes the last digit in the mantissa to be chopped off as the arithmetic unit in our hypothetical computer can accommodate only a 4-digit mantissa. Operands after the shift are as shown below:

$$\begin{array}{c} 0.5433E7 \\ 0.0045E7 \\ \hline 0.5478E7 \\ \hline \end{array}$$

Thus the sum is 0.5478E7.

EXAMPLE 2.3

Add 0.4546E3 to 0.5433E7. In this case the first operand will be shifted 4 places to the right and 0.0000E7 will be added to 0.5433E7.

EXAMPLE 2.4

Add 0.6434E3 to 0.4845E3. In this case the exponents are equal. When the mantissas are added the sum is 1.1279E3. As the mantissa has 5 digits and is greater than 1 it is shifted right one place before it is stored. When it is shifted the exponent is increased by one and the last digit of the mantissa is chopped off. Thus the answer would be 0.1127E4.

EXAMPLE 2.5

Add 0.6434E99 to 0.4845E99. In this case again the sum of the mantissas exceeds 1. Thus the mantissa is shifted right and exponent increased by 1 resulting in a value of 100 for the exponent. As the exponent part cannot store more than two digits, in our hypothetical computer, the number is larger than the largest number that can be stored in a word. This condition is called an *overflow* condition and the arithmetic unit will intimate an error condition.

Subtraction of floating point numbers

2.3.2 Subtraction

The operation of subtraction in nothing but adding a negative number. Thus the principles are the same. A few examples will now be considered to clarify some points.

EXAMPLE 2.6

Subtract 0.9432E–4 from 0.5452E–3. As the exponents are not equal the number with the smaller exponent is shifted right and the exponent increased by 1 for each right shift. Thus the result is 0.5452E–3 –0.0943E–3 = 0.4509E–3.

EXAMPLE 2.7

Subtract 0.5424E3 from 0.5452E3. The exponents are equal. Thus the mantissas are subtracted. The result is 0.0028E3. As the most significant digit of the mantissa is 0 the mantissa is shifted *left* till the most significant digit is non-zero. (Remember that in normalized floating point the mantissa is ≥ 0.1 and < 1). For each left shift of the mantissa the exponent is reduced by 1. Thus the result is 0.2800E1. The trailing zeros in the results do not carry any information but are carried by the computer in all further calculations as though significant.

EXAMPLE 2.8

The subtraction 0.5452E –99 –0.5424E –99 leads to the answer 0.0028E –99. Again the mantissa is shifted left (for normalization). In this process the exponent is reduced by 1. The exponent would thus become – 100 with the first left shift. As the exponent in our hypothetical computer can store only two digits –100 cannot be accommodated in the exponent part of the number. In this case the result is smaller than the smallest number which could be stored in this (hypothetical) computer. This condition is called an *underflow* condition and the arithmetic unit will signal an error condition.

To recapitulate, if the result of an arithmetic operation gives a number smaller than 0.1000E –99 then it is called an underflow condition. Similarly any result greater than 0.9999E99 leads to an overflow condition.

Multiplication of floating point numbers

2.3.3 Multiplication

Two numbers are multiplied in the normalized floating-point mode by multiplying the mantissas and adding the exponents. After the multiplication of the mantissas the result mantissa is normalized as in addition/subtraction operation and the exponent appropriately adjusted. Some examples are given below to illustrate the procedure.

EXAMPLE 2.9

$$+0.5543\text{E}12 \times 0.4111\text{E}-15 = 0.2278\underbrace{7273}_{\text{Discarded}}\text{E}-3 = 0.2278\text{E}-3$$

EXAMPLE 2.10

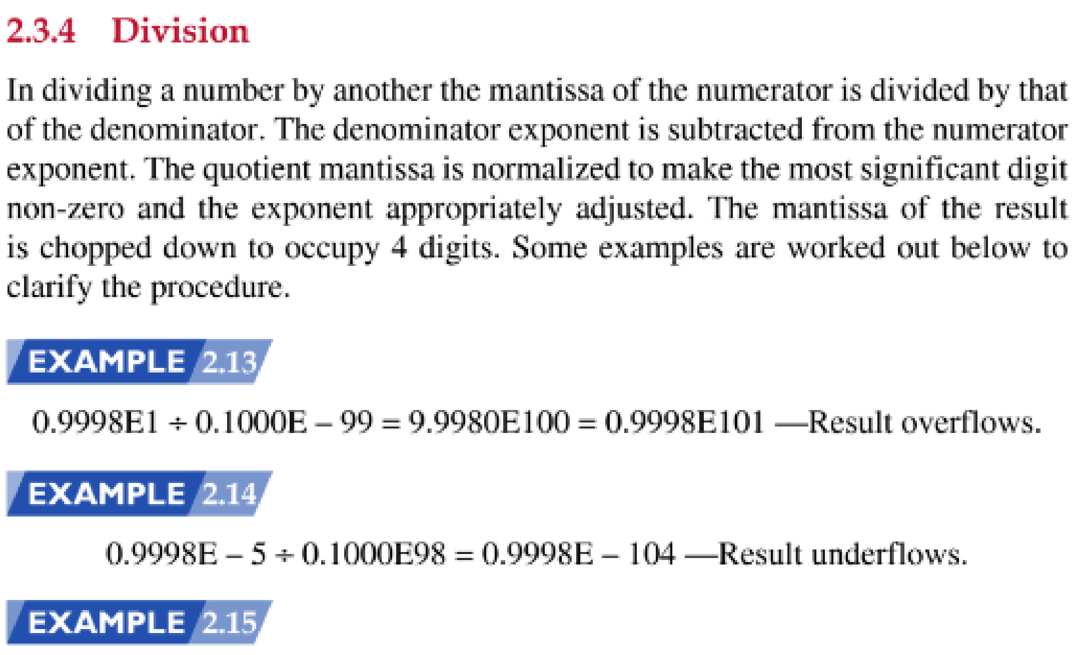
$$0.1111\text{E}10 \times 0.1234\text{E}15 = 0.0137\underbrace{0974}_{\text{Discarded}}\text{E}25 = 0.1370\text{E}24$$

EXAMPLE 2.11

$$0.1111\text{E}51 \times 0.4444\text{E}50 = 0.04937284\text{E}101 = 0.4937\text{E}100$$

—Answer overflows.

Division of floating point numbers



2.3.4 Division

In dividing a number by another the mantissa of the numerator is divided by that of the denominator. The denominator exponent is subtracted from the numerator exponent. The quotient mantissa is normalized to make the most significant digit non-zero and the exponent appropriately adjusted. The mantissa of the result is chopped down to occupy 4 digits. Some examples are worked out below to clarify the procedure.

EXAMPLE 2.13

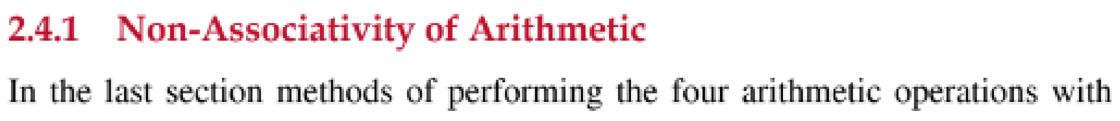
0.9998E1 ÷ 0.1000E – 99 = 9.9980E100 = 0.9998E101 —Result overflows.

EXAMPLE 2.14

0.9998E – 5 ÷ 0.1000E98 = 0.9998E – 104 —Result underflows.

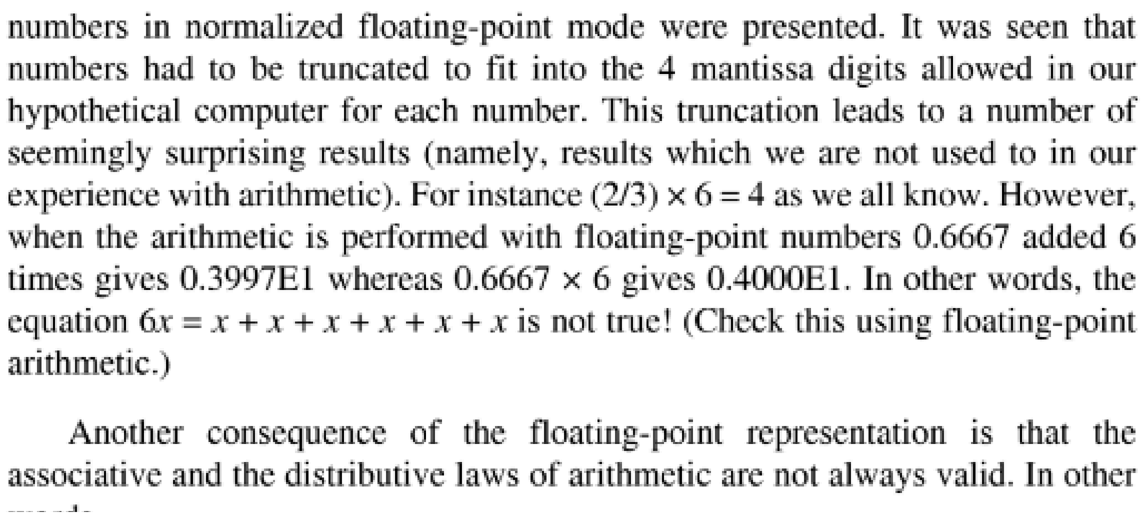
EXAMPLE 2.15

Consequences of floating point numbers



2.4.1 Non-Associativity of Arithmetic

In the last section methods of performing the four arithmetic operations with



numbers in normalized floating-point mode were presented. It was seen that numbers had to be truncated to fit into the 4 mantissa digits allowed in our hypothetical computer for each number. This truncation leads to a number of seemingly surprising results (namely, results which we are not used to in our experience with arithmetic). For instance (2/3) × 6 = 4 as we all know. However, when the arithmetic is performed with floating-point numbers 0.6667 added 6 times gives 0.3997E1 whereas 0.6667 × 6 gives 0.4000E1. In other words, the equation $6x = x + x + x + x + x + x$ is not true! (Check this using floating-point arithmetic.)

Another consequence of the floating-point representation is that the associative and the distributive laws of arithmetic are not always valid. In other

Words

$$(a + b) - c \neq (a - c) + b$$

$$a(b - c) \neq (ab - ac).$$

These are illustrated with examples below.

EXAMPLE 2.16

Let

$$\begin{aligned}
a &= 0.5665\text{E}1 \\
b &= 0.5556\text{E}-1 \\
c &= 0.5644\text{E}1 \\
(a + b) &= 0.5665\text{E}1 + 0.5556\text{E}-1 \\
&= 0.5665\text{E}1 + 0.0055\text{E}1 \\
&= 0.5720\text{E}1 \\
(a + b) - c &= 0.5720\text{E}1 - 0.5644\text{E}1 \\
&= 0.0076\text{E}1 \\
&= 0.7600\text{E}-1 \\
(a - c) &= 0.5665\text{E}1 - 0.5644\text{E}1 \\
&= 0.0021\text{E}1 = 0.2100\text{E}-1 \\
(a - c) + b &= 0.2100\text{E}-1 + 0.5556\text{E}-1 \\
&= 0.7656\text{E}-1
\end{aligned}$$

It is thus seen that $(a + b) - c \neq (a - c) + b$.

In fact the correct answer if no number is truncated is .7656E – 1.

EXAMPLE 2.17

Let

$$\begin{aligned}
a &= 0.5555\text{E}1 \\
b &= 0.4545\text{E}1
\end{aligned}$$

$$\begin{aligned}
c &= 0.4535\text{E}1 \\
(b-c) &= 0.0010\text{E}1 = 0.1000\text{E}-1 \\
a(b-c) &= 0.5555\text{E}1 \times 0.1000\text{E}-1 \\
&= 0.0555\text{E}0 = 0.5550\text{E}-1 \\
ab &= 0.5555\text{E}1 \times 0.4545\text{E}1 = 0.2524\text{E}2 \\
ac &= 0.5555\text{E}1 \times 0.4535\text{E}1 = 0.2519\text{E}2 \\
ab-ac &= 0.0005\text{E}2 = 0.5000\text{E}-1
\end{aligned}$$

Thus $a(b-c) \neq ab - ac$.

In fact, if the intermediate results are not truncated, the correct answer is 0.5555E – 1.

The above examples are intentionally chosen to illustrate the inaccuracies that may build up due to shifting and truncation of numbers in arithmetic operations. The wide disparity in the results obtained in the examples is due to the fact that in each case the difference of two almost equal numbers is involved. Whenever such a condition occurs in practice one should be careful. If possible, subtraction operation should be eliminated altogether as illustrated in the following example.

EXAMPLE 2.18

Evaluate $(1 - \cos x)$ at $x = 0.1396$ radians

$$\begin{aligned}
\cos(0.1396) &= 0.9903 \\
1 - \cos(0.1396) &= 0.1000\text{E}1 - 0.9903\text{E}0 \\
&= 0.1000\text{E}1 - 0.0990\text{E}1 = 0.1000\text{E}-1.
\end{aligned}$$

If we rewrite $(1 - \cos x)$ as $2 \sin^2 x/2$ then the value is:

$$\sin\left(\frac{x}{2}\right) = \sin 0.0698 = 0.6974\text{E}-1$$

$$\begin{aligned}
2\sin^2\frac{x}{2} &= (0.2000\text{E}1) \times (0.6974\text{E}-1) \times (0.6974\text{E}-1) \\
&= 0.9727\text{E}-2
\end{aligned}$$

The value obtained by the alternate formula is closer to the true value 0.9728E – 2.

Concept of ZERO in floating point arithmetic

The number zero has a precise meaning in arithmetic. However, while doing arithmetic with real numbers represented in normalized floating-point mode exact equality of a number to zero can never be ensured. This is again due to the fact that most numbers in floating point are only approximations. To fix our idea we will consider an example.

EXAMPLE 2.20

The roots of the quadratic equation $x^2 + 2x - 2 = 0$ are $-1 \pm \sqrt{3}$. The roots expressed in floating point with 4-digit mantissa are 0.7320E0 and –0.2732E1. If we substitute 0.7320E0 for x in the expression $(x^2 + 2x) - 2$ we should ideally obtain zero. Using floating-point arithmetic, however, we get

$$(0.7320\text{E}0 \times 0.7320\text{E}0 + 0.2000\text{E}1 \times 0.7320\text{E}0) - 0.2000\text{E}1$$

$$= (0.5358\text{E}0 + 0.1464\text{E}1) - 0.2000\text{E}1 = -0.1000\text{E} - 2$$

If instead of computing $(x^2 + 2x)$ first and then subtracting 2 from it, we compute $(2x - 2)$ and add it to x^2 (using floating-point arithmetic) we get –0.2000E – 3! Similarly, if –0.2732E1 is substituted for x in $(x^2 + 2x) - 2$ we get –0.1000E – 2.

Thus even though the roots are substituted in the quadratic expression the result is not zero. *The main point to be noted is that in any computational algorithm it is not advisable to give a branching instruction based on testing whether a floating-point quantity is zero or not zero.*

In the previous chapter an algorithm was formulated to solve the simultaneous equation

$$ax + by = c$$

$$px + qy = r.$$

a	b	c	p	q	r
2.500	5.200	6.200	1.251	2.605	3.152

If the equations are solved using Algorithm 1.2 and ordinary arithmetic (keeping all the digits obtained in intermediate computations) the solution would be:

$$x = -0.32794\text{E}2 \qquad y = 0.16958\text{E}2$$

Using floating-point arithmetic with 4-digit mantissa if Algorithm 1.2 is executed the solution obtained would be:

$$x = -0.3217\text{E}2 \qquad y = 0.1666\text{E}2$$

(The student is urged to check this using floating-point arithmetic before proceeding further.)

Observe that the two solutions agree only in the two most significant digits. The reason for the disagreement in the two solutions is again due to loss of significant digits when the difference of two almost equal numbers is taken. The solutions obtained will be equal in this particular case if 8 digits rather than 4 digits are used for the mantissa.

A still more startling result would be obtained in this example if the value of q is changed to 2.606 and the values of all the other variables are kept the same. Observe that this is a change of 0.001 in one of the numbers. The solution obtained for x and y using 4-digit mantissa and floating-point arithmetic is: $x = -0.2352\text{E}2$ and $y = 0.1250\text{E}2$. A change in one coefficient by less than 1 in two thousand has led to a change of more than 30 per cent in the values of x and y! This coefficient may have been obtained from other computations with

Assesment

1.1 Revise the instructions for solving the simultaneous equations given at the beginning of the chapter for the case when one or more of the coefficients a, b, c, p, q, r are zero.

1.2 In the program for solving simultaneous equations what is the physical significance of $(aq - bp)$ being zero?

1.3 Solve the simulataneous equations with the following values of a, b, c, p, q and r.

$a = 2.500$	$b = 5.200$	$c = 6.200$
$p = 1.251$	$q = 2.605$	$r = 3.152$

Give the answer correct to four decimal digits.

1.4 Obtain an algorithm to find the sum of the following series to the first n terms

(i) $\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots$

(ii) $\log_e(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

1.5 Obtain an algorithm to compute the sum S

$$S = \frac{1}{px + q}\sum_{i=0}^{n} a_i x^i$$

1.6 Obtain an algorithm to evaluate

$$\frac{dp(x)}{dx}$$

where $p(x)$ is the nth degree polynomial

$$(a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0)$$

1.7 Develop an algorithm to read a vector, sum all its positive components, count such components and sum separately all the negative components of the vector.

1.8 Obtain an algorithm which given the coordinates of a point (x, y) will write a message whether it is in or not in the first quadrant of the unit circle.