Chapter 1

□ INTRODUCTION

1.1 Hardware and Software

A computer is a combination of electrical and mechanical parts called *Hardware* along with the instructions (programs) that direct the operation of the computer. These instructions are called *Software.*

1.1.1 Components of a Computer

The main components of a computer are (i) *Input-Output* devices, (ii) *Central Processing* units, and (iii) *Storage* devices.

Input Devices These allow us to put information or instructions into the computer. The most common input device today is the *keyboard.* Other input devices , used for special applications are, *optical scanners* of various types, e.g., Bar-Code readers, Optical Mark Readers, Facsimile (FAX) machines, and Telephone lines.

Output Devices The allow us to get and see from the computer. The primary output device is the *TV monitor.* The printer is also a common output device. Other output devices are graphics plotters, photo-typesetters, etc.

Storage Devices : Data and instructions are stored in the computers memory which consists of many very simple *two-state* devices. These devices are either *ON* or *OFF.* A light bulb is a two-state device.

The *primary memory* of a computer can be viewed as a set of boxes, with each box having a unique number called an *address.* The *secondary memory* usually consists of *disks drives* of some form, e.g., *floppy*, *hard*, *compact*, *optical.* In older computers *magnetic tapes* are used.

Central Processing Unit The CPU is the heart of the computer. It contains two parts. The *Arithmetic-Logic* unit performs (1) *Addition*, (2) *Subtraction*, (3) *Multiplication*, (4) *Division*, and (5) *Comparison/Jump.*

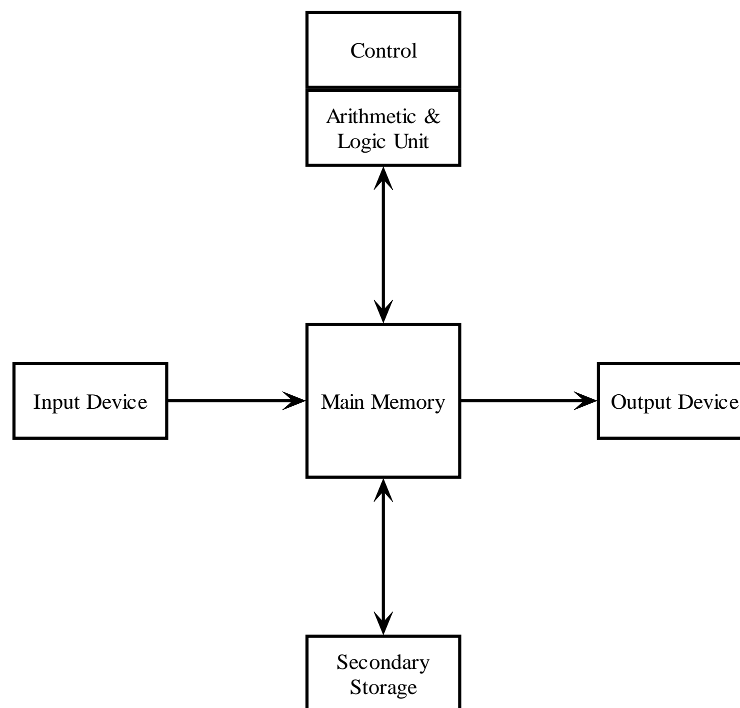

Figure 1.1. Components of a Computer.

The *Control Unit* directs the operation of the entire computer system by interpreting and executing instructions from the keyboard and memory.

1.1.2 Software

Software consist of the *instructions*, in the form of programs, that get the computer to do something useful. The primary piece software in any computer is the *Operating System*. This is a program that is always running when the computer is turned on and it directs the operation of all hardware and other programs.

Applications Software are programs that do special jobs. Some typical examples are *Programming Language Compilers*, *Word Processors*, *SpreadSheets*, *Data-Base Systems*, *Typesetters*, *Graphics*, *Accounting Systems*.

1.2 Information representation

All information used by a computer must be represented using two-state devices. Such a device is said to contain 1 *Bit* of information. The primary memory consists of bits grouped into *Bytes* of 8 bits and these bytes are further grouped into *Words* of 2, 4, or 8 bytes. Standard micro-computers today have memories of 4–256MBytes. A type-written page can be stored in 1–2KBytes.

A byte of 8 bits can represent $2^8 = 256$ different things and a word of 4 bytes or 32 bits can represent $2^{32} = 4,294,967,296$ different things.

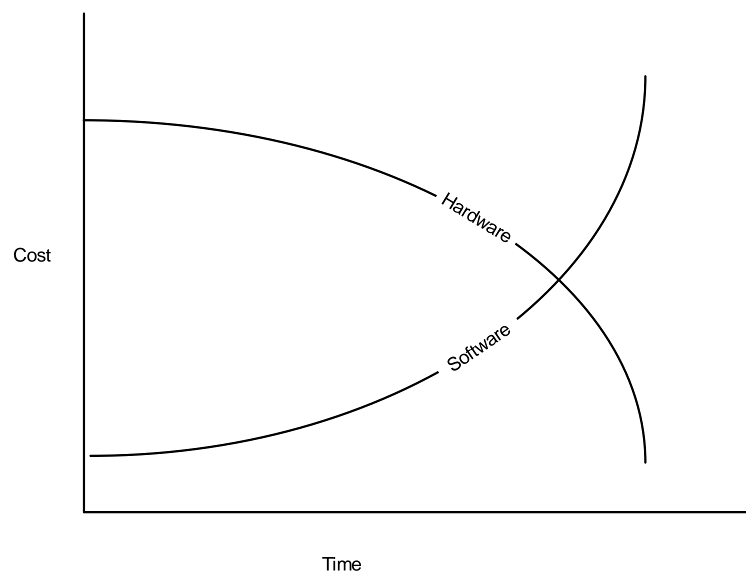

Figure 1.2. Hardware and Software Costs.

1.2.1 Number Systems

The decimal number 426.23 is an example of a *positional number system*

$$426.23 = 4 \times 10^2 + 2 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}.$$

The *Decimal* number system uses 10 symbols : $\{0, 1, \ldots, 9\}$.

The *Octal* number system uses 8 symbols : $\{0, 1, \ldots, 7\}$.

The *Binary* number system uses 2 symbols : $\{0, 1\}$.

EXAMPLE : *Binary Numbers.* The binary number $(1001101)_2$ in expanded form is

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 64 + 0 + 0 + 8 + 4 + 0 + 1 = (77)_{10}.$$

The fractional binary number $(1001.101)_2$ in expanded form is

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$= 8 + 0 + 0 + 1 + 0.5 + 0 + 0.125 = (9.625)_{10}.$$

□

1.2.2 Storage of Numbers, Characters and Instructions

Computer memory consists of a set of boxes (bytes) each of which can contain 8 bits. To store an integer number in a byte we allow the first bit to represent the sign of the number : $0 = +$

and 1 = −. The remaining 7 bits are used to represent the magnitude of the number. This means we can represent $2^7 = 128$ different magnitudes along with their signs in an 8-bit byte. Figure 1.3 shows the storage of the number $-(1000101)_2$

$$-(1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 64 + 0 + 0 + 0 + 4 + 0 + 1 = -(69)_{10}.$$

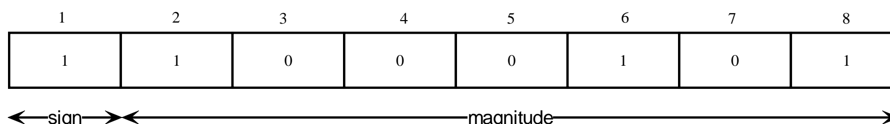

Figure 1.3. Integer Storage in a Byte.

To store numbers of larger magnitude we use 2 bytes or 4 bytes in combination.

To store alphabetic and other keyboard characters we first transform the characters into integers between 0 and 255 and then store these decimal integers as binary numbers. A standard transformation is the *ASCII* code : A = 65, B = 66, ..., Z = 90, etc.

A computer usually has a small set of *Hardware Instructions* that are performed by the control unit along with the arithmetic/logic unit. Some of these instructions are : ADD, LOAD, STORE, MOVE. Long sequences of these simple instructions are used to perform more complicated tasks. As with numbers and characters, these instructions are transformed into binary numbers and stored in memory.

Thus we see that all instructions and data are ultimately stored as strings of 0's and 1's, i.e., strings of bits.

1.3 Algorithms

The study (design and analysis) of algorithms is central to all areas of Computer Science. The word comes from the Persian mathematician *M. al-Khowarizmi* who lived in the 9th century.

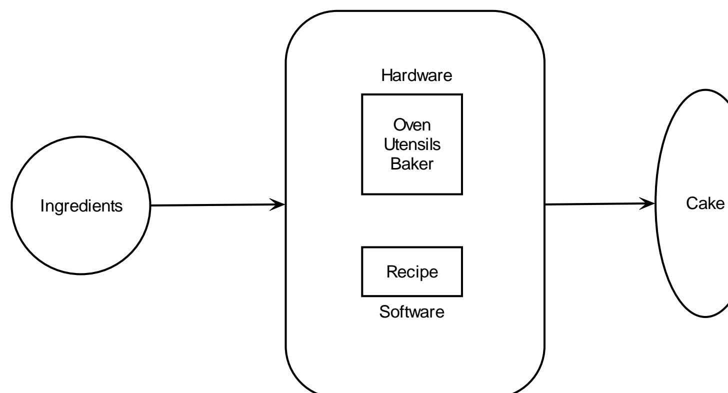

Figure 1.4. Baking a Cake.

Algorithm. A finite set of steps required to perform some task in a finite amount of time.

EXAMPLE : *Changing a Wheel.*

algorithm *ChangeWheel(INPUT : Spare Wheel.*
OUTPUT : ?
HARDWARE : Jack, handle,driver.

```
[Step1]   Get spare wheel, jack, and handle

[Step2]   Jack up car

[Step3]   Remove flat wheel

[Step4]   Put on spare wheel

[Step5]   Jack down car

[Step6]   Put flat wheel, jack and handle in boot
```

□

Levels of Detail Depends on 'intelligence' of hardware. Algorithm must be tailored to fit the hardware's capabilities.

EXAMPLE : *Finding the Maximum.* Given a deck of cards, each with a number printed on it, find the card with the maximum number.

algorithm *Maximum(INPUT : 100 cards each*
with a 4-digit number.
OUTPUT : Card with largest number.
HARDWARE : You.

```
[Step1]   Pick up first card

[Step2]   Pick up next card

[Step3]   Throw away card with smaller number

[Step4]   Repeat Steps 2 to 4 until no cards left to pick up

[Step5]   The card remaining in your hand is has max.  number
```

□

This algorithm introduces a new idea : *repetition* in Step 4. Repetition allows us to write very short algorithms for long processes. The algorithm works for any finite number of cards. (In fact this algorithm implicitly introduces another idea which we'll see in the next example)

EXAMPLE : *Is a List Sorted ?* Given a deck of cards, each with a name printed on it, determine if the deck is sorted into ascending order.

algorithm *Sorted(INPUT : A deck of cards each with a name.*
OUTPUT : Yes or No.
HARDWARE : You.

```
[Step1]   Pick up first card in left hand

[Step2]   Pick up next card in right hand

[Step3]   if LH card > RH card then STOP with answer NO
              else continue to next step

[Step4]   Throw away LH card

[Step5]   Transfer RH card to LH

[Step6]   Repeat Steps 2 to 6 until no cards left

[Step7]   STOP with answer YES
```

We notice in this algorithm that we have used another new idea : a *decision* or *selection* in [Step 3]. □

1.3.1 Control Structures

These allow us to control the order in which steps of an algorithm are carried out. The standard control structures are *Sequence*, *Selection*, and *Repetition*

Combining Control Structures Control structures can be viewed as big steps that can be combined to make larger and more complicated algorithms.

EXAMPLE : *BubbleSort.* Sort n numbers into ascending order. Assume the numbers are stored in an array of n contiguous boxes as shown.

1	2	3	4	5	6	7	8	9	10	11	12
45	23	51	76	12	81	32	38	64	15	37	17

Figure 1.6. n unsorted numbers.

-

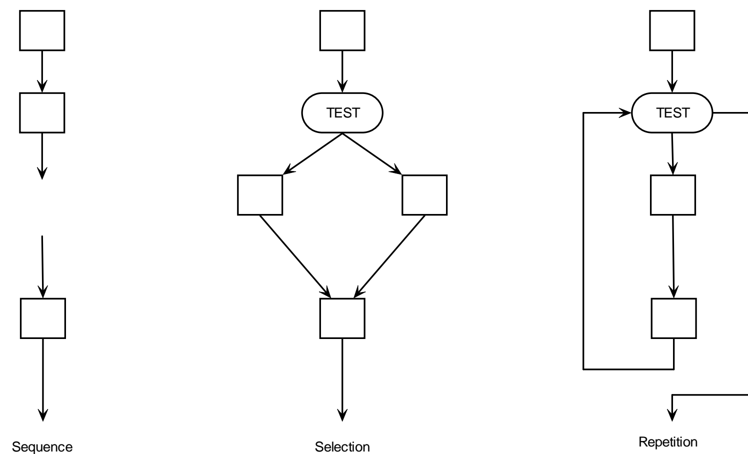

Figure 1.5. Control Structures.

algorithm *BubbleSort(INPUT : an array of unsorted numbers.*
OUTPUT : an array of sorted numbers.)

```
[Step 1] Do the following n − 1 times
    [Step 1.1] Point to the first number
    [Step 1.2] Do the following n − 1 times
        [Step 1.2.1] compare number with next number
        [Step 1.2.2] if in wrong order, switch
        [Step 1.2.3] point to next number
```

□

The number of times the inner-most instructions are performed is

$$(n-1) \times (n-1) = n^2 - 2n + 1.$$

Question : Can the BubbleSort algorithm be made more efficient? □

1.3.2 Functions

Functions are useful if we need to do the same work on different but similar sets of inputs (ingredients).

Calculating the Square Root of a Number.

This is a task that is performed many times in geometric and scientific calculations. Although many calculators have a $\surd$ button, behind the button is an algorithm that calculates the square root.

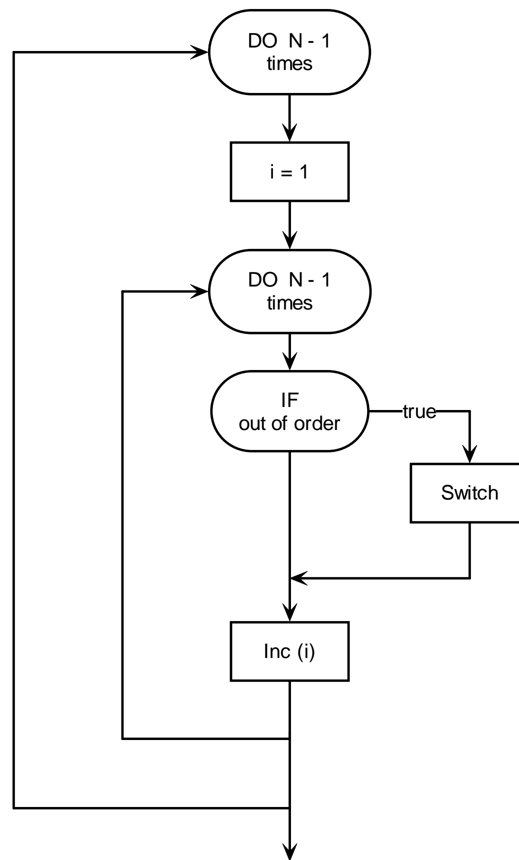

Figure 1.6. Structures of BubbleSort.

algorithm *Sqrt (Input : N; Output :* $\sqrt{N}$*)*

[Step 0] $X_{old} = N/2$ --- initial guess.
[Step 1] Do the following until satisfied
  [Step 1.1] $X_{new} \Leftarrow \frac{1}{2} \times (X_{old} + N/X_{old})$
  [Step 1.2] $X_{old} \Leftarrow X_{new}$
[Step 2] Return $X_{new} \approx \sqrt{N}$

Note : The hardware is assumed to able to do the operations $+$, $\times$, $/$, and $\Leftarrow$.

EXAMPLE : *The Square Root of $N = 2$*

X_{old}	$\frac{1}{2}(X_{old} + N/X_{old})$	$= X_{new}$
1.000000	0.5(1.000000 + 2/1.000000)	= 1.500000
1.500000	0.5(1.500000 + 2/1.500000)	= 1.416667
1.416667	0.5(1.416667 + 2/1.416667)	= 1.414157
1.414157	0.5(1.414157 + 2/1.414157)	= 1.414214
1.414214	0.5(1.414214 + 2/1.414214)	= 1.414214

□

This algorithm *Returns* a value and for this reason it is called a *Function.*The algorithm was invented by Sir Isaac Newton in 1650 and is still in use today in all computers and calculators. Although tedious for humans it is ideally suited to digital computers.

We can now use this function as if it were an additional operation built into the hardware. Hence we can use it in any other algorithm. For example, suppose we wish to calculate

$$\sqrt{(a^2 + b^2)} + 2.0 + \sqrt{x}.$$

The algorithm steps are as follows :

[Step 0]	Get values for a, b, x
[Step 1]	$R1 \Leftarrow a \times a + b \times b$
[Step 2]	$R2 \Leftarrow \mathbf{Sqrt}(R1)$
[Step 3]	$R3 \Leftarrow R2 + 2.0$
[Step 4]	$R4 \Leftarrow \mathbf{Sqrt}(x)$
[Step 6]	$R6 \Leftarrow R3 + R4$

1.3.3 Recursion

Recursion is a very powerful concept that is fundamental to mathematics and computer science. One of the corner-stones of mathematics is *Recursive Function Theory.* In computer science procedures or algorithms that can call or *use themselves* are often used to build elegant and efficient algorithms that solve many practical computing problems.

Recursive Functions These are functions whose definition refer to themselves, e.g.,

$$N! = N \times (N-1)!, \quad (N-1)! = (N-1) \times (N-2)!$$

An algorithmic definition of this function is as follows :

function Fact(N)
if $N = 1$ **then** $Fact(N) = 1$ **else** $Fact(N) = N \times Fact(N-1)$

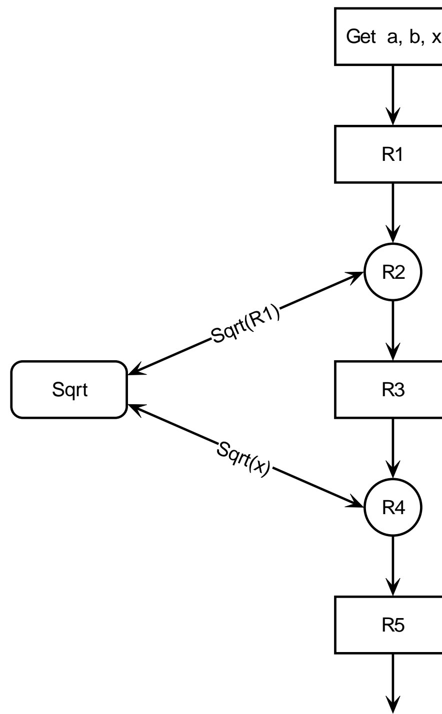

Figure 1.7. Algorithm Structure using Functions.

For example, if $N = 4$ then we have

$$\begin{aligned}
&Fact(4) = 4 \times Fact(3)\\
&\quad Fact(3) = 3 \times Fact(2)\\
&\qquad Fact(2) = 2 \times Fact(1)\\
&\qquad\quad Fact(1) = 1\\
&\qquad Fact(2) = 2 \times 1 = 2\\
&\quad Fact(3) = 3 \times 2 = 6\\
&Fact(4) = 4 \times 6 = 24
\end{aligned}$$

Integer Arithmetic using Recursive Functions

Integer powers of a number can be defined recursively as $x^N = x \times x^{N-1}$. The algorithm for this function is

function `Pow`(x, N)

---

if $N = 0$ **then** `Pow`$(x, N) = 1$
else `Pow`$(x, N) = x \times$ `Pow`$(x, N - 1)$

Recursive functions work by breaking a given problem into smaller and smaller pieces until a trivial problem results.

EXAMPLE : *Maximum of an array of N numbers.* The key idea in the recursive solution of this problem is to break the array into two, solve the two smaller problems recursively and the combine their results as follows :

MaxArray $(1, N)$ = Max (MaxArray(Left Half),MaxArray(Right Half))

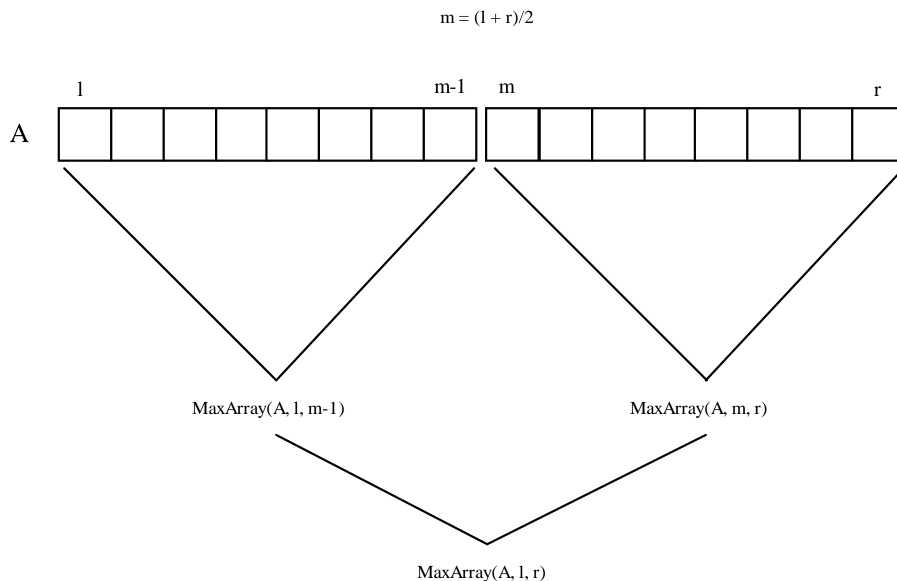

Figure 1.8. Recursive MaxArray, .

The algorithm for this function is

```
function MaxArray(A, l, r)
---------------------------------------------
    if l = r THEN MaxArray = A[l]
    else
     m = (l + r)/2
     MaxArray = Max {MaxArray(A, l, m − 1),MaxArray(A, m, r) }
```

□

It should be noted that there is no repetition control structure in this algorithm, yet all elements of the array are examined. This shows that *recursion is an alternative to repetition.*

1.3.4 The Towers of Hanoi

This is an ancient puzzle which is said to have originated in a monastery in Tibet. The monks were required to move 64 gold rings from one diamond peg to another as shown. It was said that it would take until the end of the world to complete the task.

The rings are to be moved from A to C using the following rules :

1. Rings moved one-at-a-time
2. A larger ring may not be placed on top of a smaller ring.
3. The peg B may be used as intermediate storage.

Let $X \to Y$ represent the general move

Move the top ring on peg X to peg Y

The general problem is

Move N rings from A to C using B

If $N = 1$ The Problem is easy : $A \to C$.

If $N = 2$ the following moves solve the problem :

$$A \to B, \quad A \to C, \quad B \to C.$$

If $N = 3$ the following moves solve the problem :

$$A \to C, \quad A \to B, \quad C \to B, \quad A \to C,$$

$$B \to A, \quad B \to C, \quad A \to C.$$

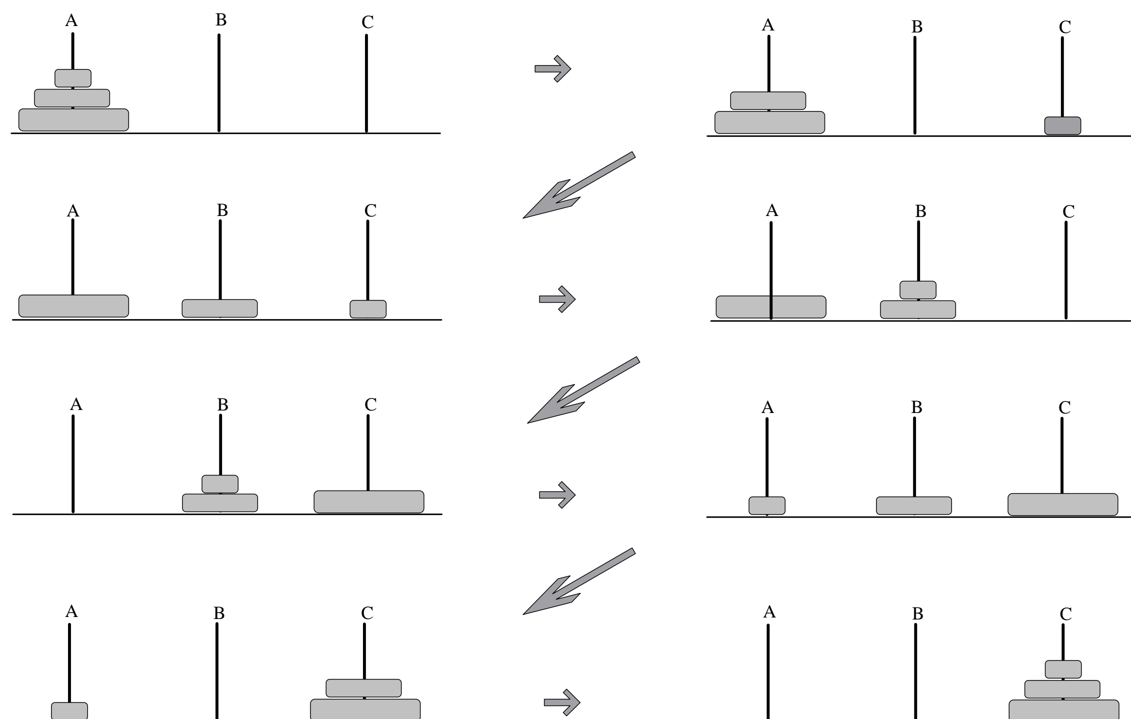

Figure 1.8. Towers of Hanoi, $N = 3$.

We now use recursion to break the general problem into smaller pieces. The key idea is the following general move for N rings :

Move $N-1$ rings from A to B; $A \rightarrow C$; *Move $N-1$ rings from B to C.*

The formal algorithm is as follows :

```
algorithm Hanoi(A, B, C, N)
------------------------------------------------
        if N = 1 then Move top disk of A to C
        else
                    Hanoi(A,C,B,N-1)
                    Move top disk of A to C
                    Hanoi(B,A,C,N-1)
         endif
     endalg Hanoi
```

1.3.5 Programs

A program is a translation of an algorithm (formal or informal) into a well defined set of instructions using the *syntax or grammar* and the *semantics or meaning* of a formal programming language.

Because we'll be spending the rest of this course on programming we will simply give examples of programs that correspond to two of the algorithms in the previous sections.

The first example is not a complete program but is the definition of the function $Sqrt(x)$ written in Pascal.

```
function Sqrt(var n : REAL) : REAL;
------------------------------------------------
    var xold, xnew : REAL;
         satisfied  : BOOLEAN;
    begin
        xold := n/2.0;
        satisfied := false;
        while NOT(satisfied) do begin
              xnew := 0.5*(xold + n/xold);
              satisfied := (Abs(xnew-xold) < 0.000001);
              xold := xnew;
        end; {WHILE}
         Sqrt := xnew;
     end; {FUNC Sqrt}
```

This is a partial or sub-program called a *function* that can be used by any other program yhat requires it.

The second example is the complete program for the Towers of Hanoi problem.

```
PROGRAM TowersOfHanoi;
   VAR A,B,C : CHAR;
       N     : INTEGER;
       count :  INTEGER;
 PROCEDURE Pause;
   BEGIN
      Write('Press <RETURN> to continue...');
      Readln;
   END; { PROC Pause }
 PROCEDURE Hanoi(VAR A,B,C : CHAR; N : INTEGER);
   BEGIN
      IF N=1 THEN BEGIN
             Inc(count);
             Writeln(count,' Move top disk of ', A, ' to ', C);
      END
      ELSE BEGIN
             Hanoi(A,C,B,N-1);
             Inc(count);
             Writeln(count,' Move top disk of ', A, ' to ', C);
             Hanoi(B,A,C,N-1);
      END;
   END; {PROC Hanoi}
   BEGIN { The main program statements begin here }
         A := 'L';
         B := 'M';
         C := 'R';
         count := 0;
         Write('Type in how many disks you want to move :');
         Readln(N);
         Hanoi(A,B,C,N);
         Pause;
   END. {PROGRAM TowerOfHanoi}
```

Laboratory Exercise No. 1: *Starting Turbo Pascal..*

Using the Turbo Pascal system, type in the program above, **exactly as given**, compile it and run it for various values of N.