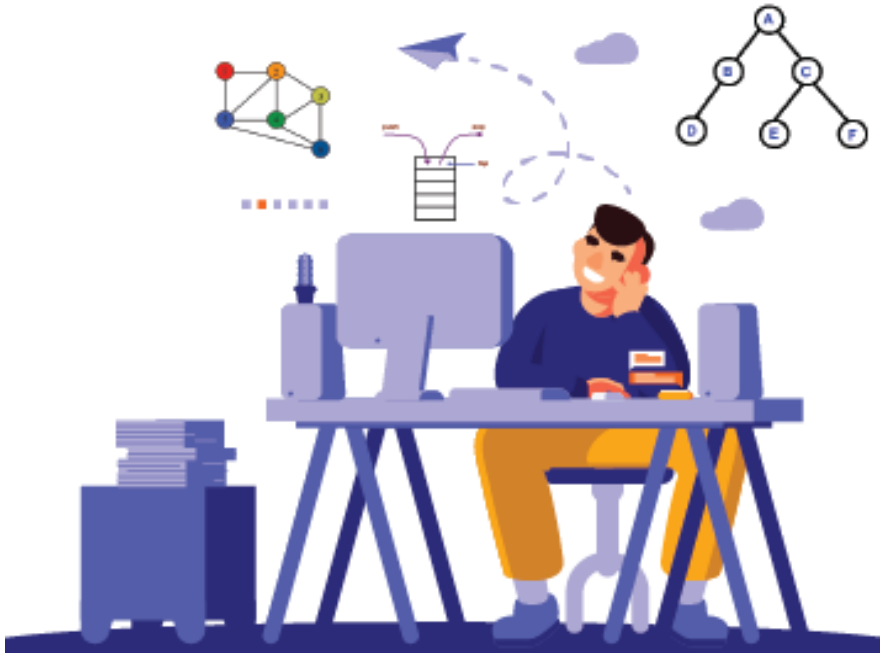
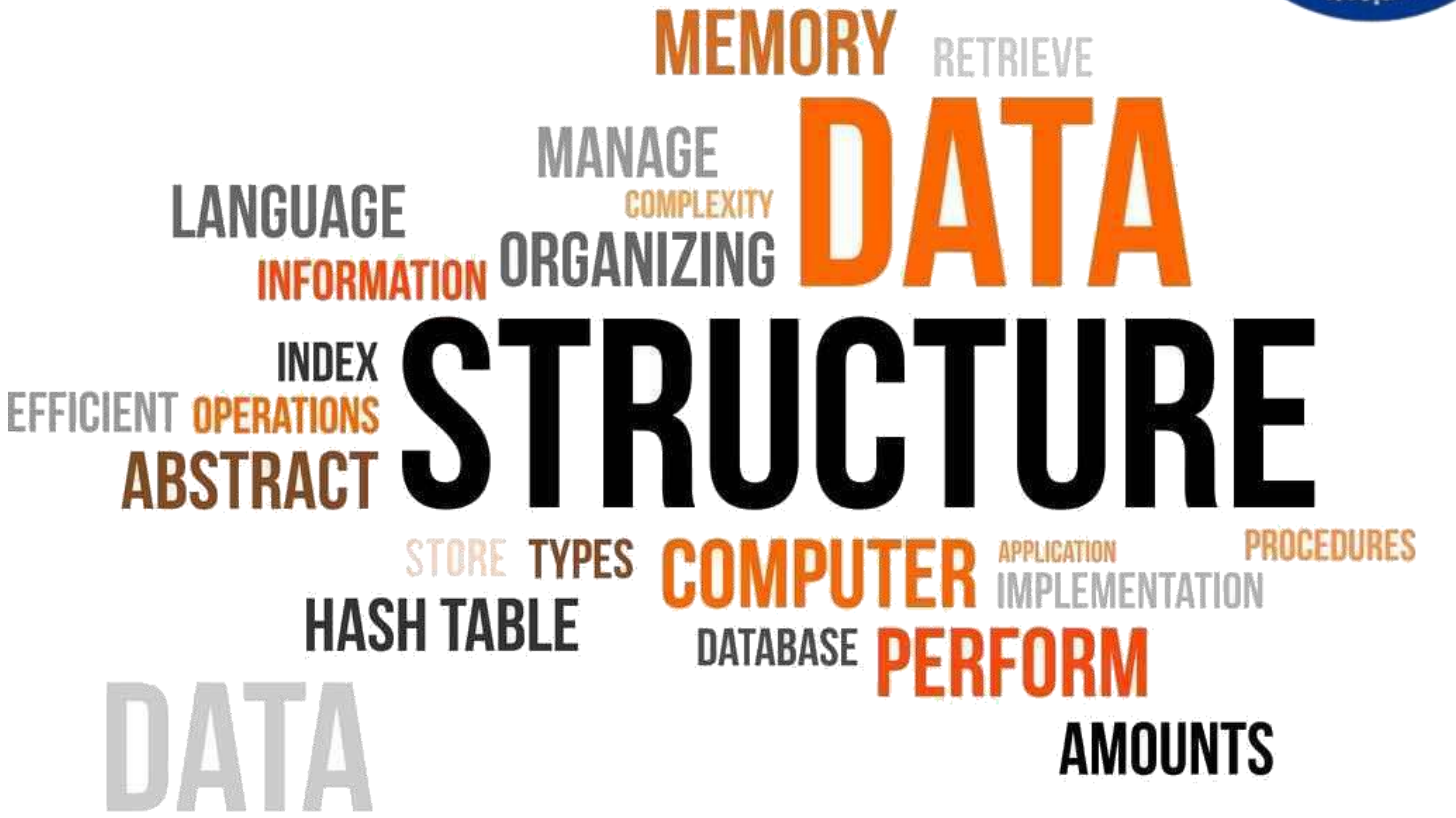


# Data Structures

**Course code: IT623**




**Dr. Rahul Mishra**
**Assistant Professor**
**DA-IICT, Gandhinagar**

# Lectures 8

## *String Processing*

# String

A **string is a sequence of characters** that can be used to represent text or data to communicate with the computer. It is a fundamental data type in many programming languages and is also used extensively in data structures and algorithms.

In data structures, strings are often represented as arrays of characters, where each character in the array corresponds to a single element in the string. Strings can also be represented as linked lists or trees, where each node in the data structure represents a single character in the string.

There are several operations that can be performed on strings in data structures, including:

1. ***Concatenation:*** Combining two or more strings into a single string.

1. ***Substring:*** Extracting a portion of a string, such as a prefix, suffix, or substring between two indices.

1. ***Comparison:*** Comparing two strings to determine if they are equal or which one is lexicographically larger.

1. ***Searching:*** Searching for a particular substring or character within a string.

1. ***Modification:*** Changing or updating the contents of a string.

# String

Algorithms that use strings and string operations include **pattern matching, string sorting, and string compression.**

String algorithms are often used in ***text processing applications, such as search engines, text editors, and spell checkers.***

The string has also termed a set of characters, including the following:

**Alphabet:** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

**Digits:** 0 1 2 3 4 5 6 7 8 9

**Special Characters:** + - / % ( ) $ % ^ ,. ‘ “

# String Processing :-

* Traditionally, we use only characters due to limited capability of the machines
* Emphasizing the primary application of computer today is in the field of word processing.
* (Pattern matching)
* Computer terminology usually use the term "string" for a sequence of characters rather than the term "word," since "word" has another meaning.
* String processing ⟶ word processing ∝
  String manipulation ⟶
  Text editing ⟶

## * Basic Terminology :

→ Each programming language contains a character set that is used to communicate with computer usually includes:

- Alphabet :- A B C D . . . X Y Z
- Digits :- 0 1 2 3 . . . 8 9
- special characters :- + − / * ( ), . $ = ' □

* The set of special characters, which includes the blank space, frequently denoted by □, varies somewhat from one language to another.

* **Length:-** The number of characters in a string is called its length.

* **Empty string :-** The string with zero characters is called the empty string or null string.

* String are denoted by enclosing their characters in single quotation marks.

* This quotation marks also serve as string delimiters

'DSA' → Length = 3

'MNC104' → Length = 6

'Asymptotic' → Length = 10

'DD' → Length = 2

' ' — empty space. → Length = 0

'Be consistent Be Extraordinary'

↳ Length = 27 or 30 ?

'BE CONSISTENT BE EXTRAORDINARY'

2 10 2 13

* We emphasize that the blank space is a character and hence contributes to the length of the string.

* Quotation marks may be omitted when the context indicates that the expression is a string.

* Let $S_1$ and $S_2$ be strings. The string consisting of the characters of $S_1$ followed by the characters of $S_2$ is called the concatenation of $S_1$ and $S_2$; it will be denoted as by $S_1 // S_2$.

* For example,

'THE' // 'END' = 'THEEND' but 'THE' // '□' // 'END' = 'THE END'

* The length of $S_1 // S_2$ is equal to the sum of the lengths of the strings $S_1$ and $S_2$.

## Sub string :

A string (Y) is called substring of a string (S) if there exist (X) and (Z) such that

$$S = X//Y//Z$$

IF X is a empty string then Y is called an initial substring of S, and if Z is an empty string then Y is called a terminal substring of S.

Example:

"BE OR NOT" is a substring of "TO BE OR NOT TO BE"

"DSA" is substring of "DSA MNC"

Try some examples for { initial substring, terminal substring }

## Storing strings :-

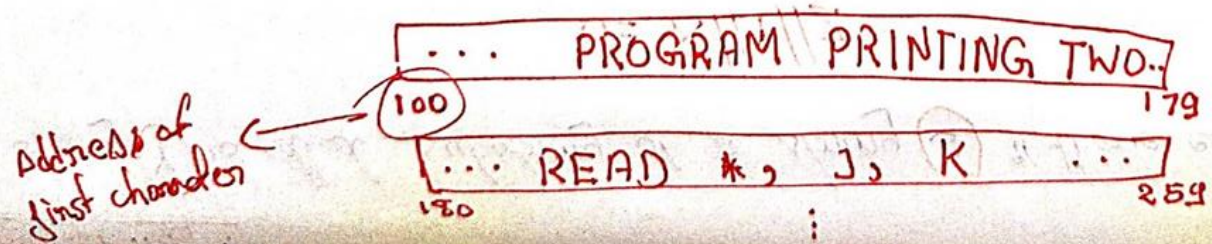
Strings are stored in three types of structure :

(1) Fixed-length structures
(2) Variable-length structures with fixed maximums
(3) Linked structures

### * Record-Oriented, Fixed-Length Storage

* In fixed-length storage each line of print is viewed as a record, where all records have the same length, i.e., where each record accomodates the same number of characters.

* Since data are frequently input on terminals with 80-column images or using 80-column cards, we will assume our records have length 80 unless otherwise stated or implied.



# Example of fixed length storage

**Fig. 3.1** *Input Data*

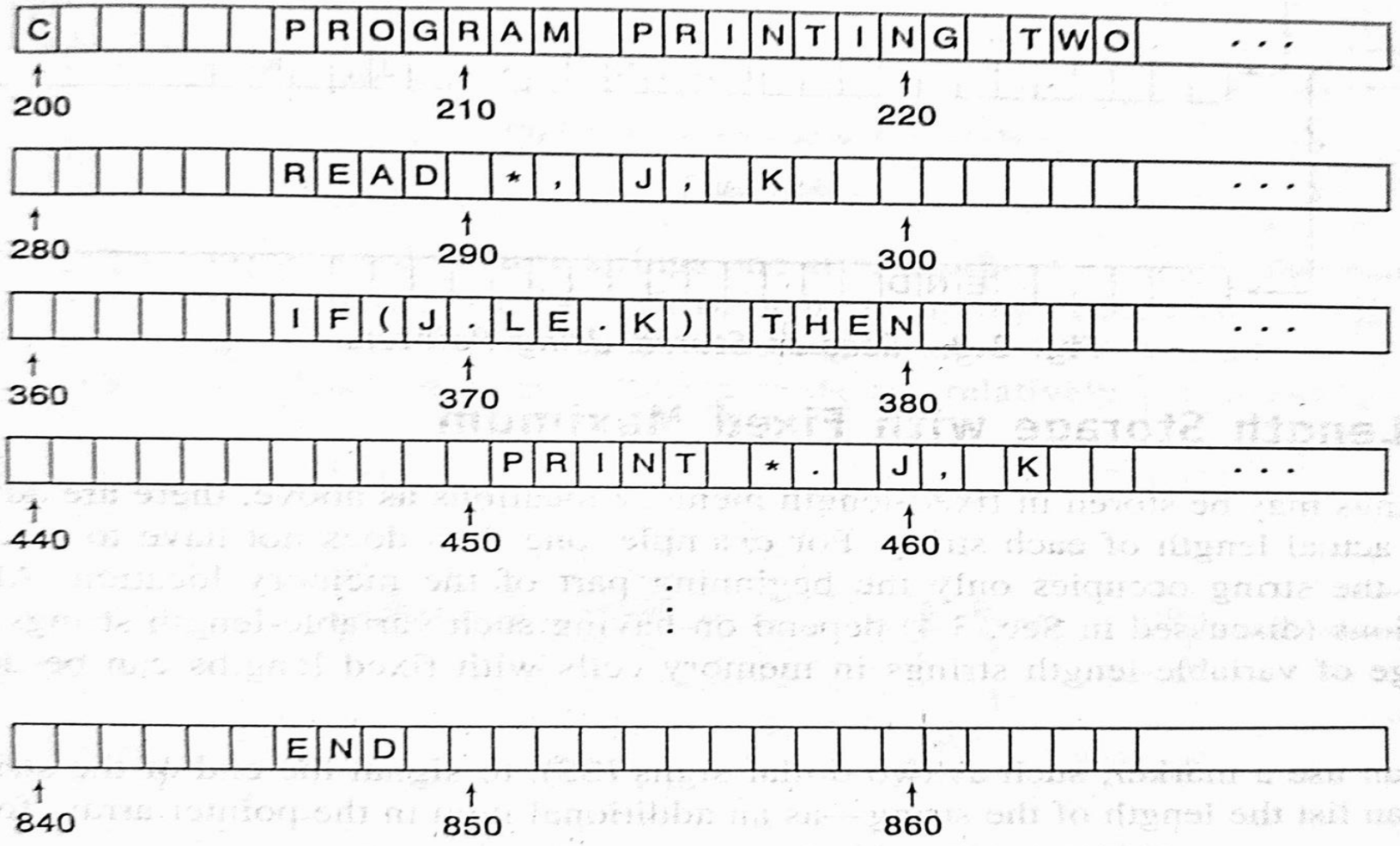



**Fig. 3.2** *Records Stored Sequentially in the Computer*

Advantages: The main advantages of the above way of sorting strings are:

① The ease of accessing data from any given record.

② The ease of updating data in any given record (as long as the length of the new data does not exceed the record length).

Disadvantages: The main disadvantages are:

(1) Time is wasted reading an entire record if most of the storage consist of inessential blank spaces.

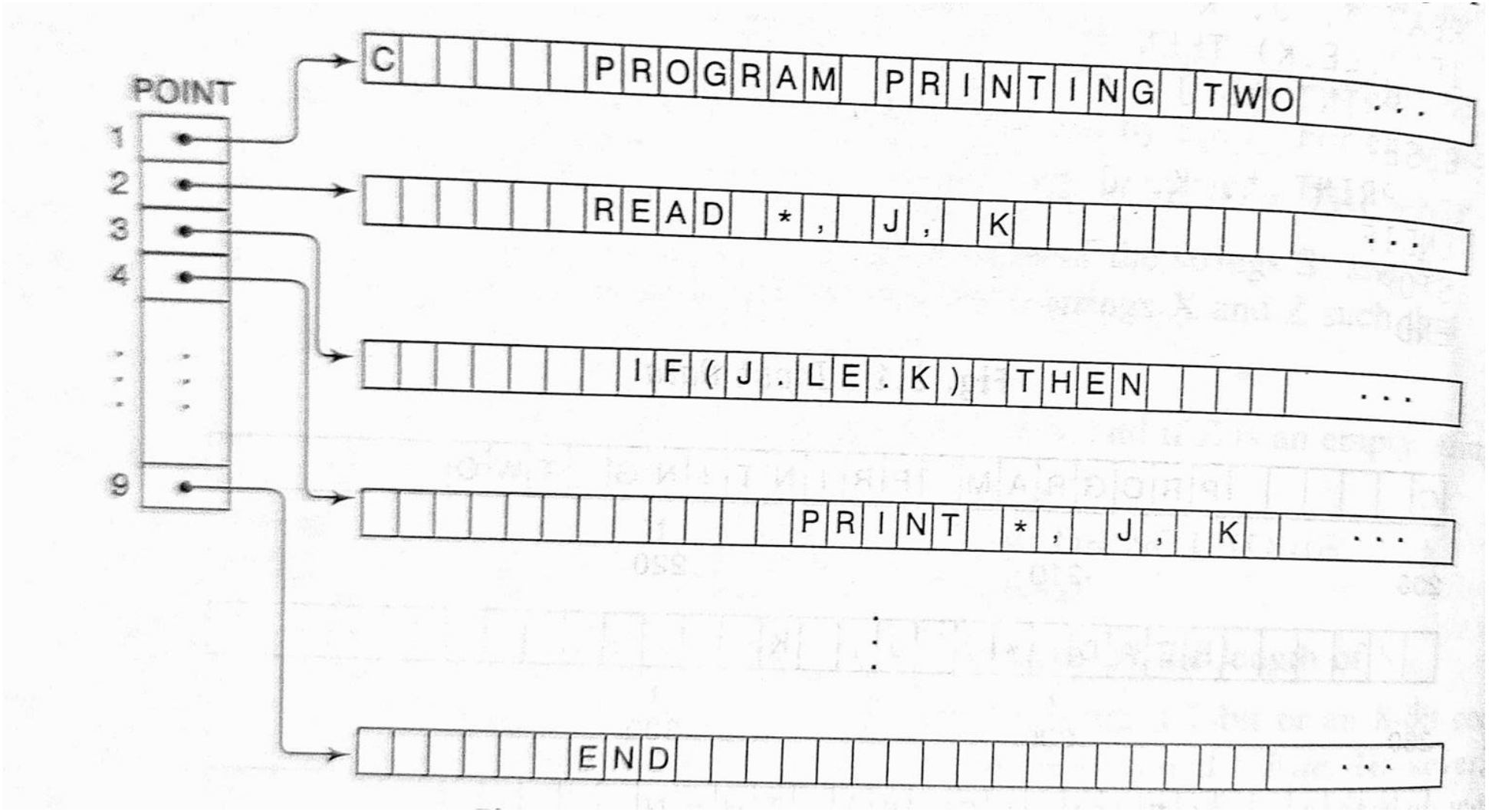
(2) Certain records may require more space than available.

(3) When the connection consists of more or fewer characters than the original text, changing a misspelled word requires the entire record to be changed.

Notably:

* Suppose we wanted to insert a new record in previous example.
* We require that all succeeding records be moved to new memory locations.
* To mitigate this disadvantage, we use following:
* One can use a linear array POINT which gives the address of each successive record.
* The records need not be stored in consecutive locations in memory.
* Inserting a knew record will require only on updating of the array POINT.

# Example of fixed length storage with pointer

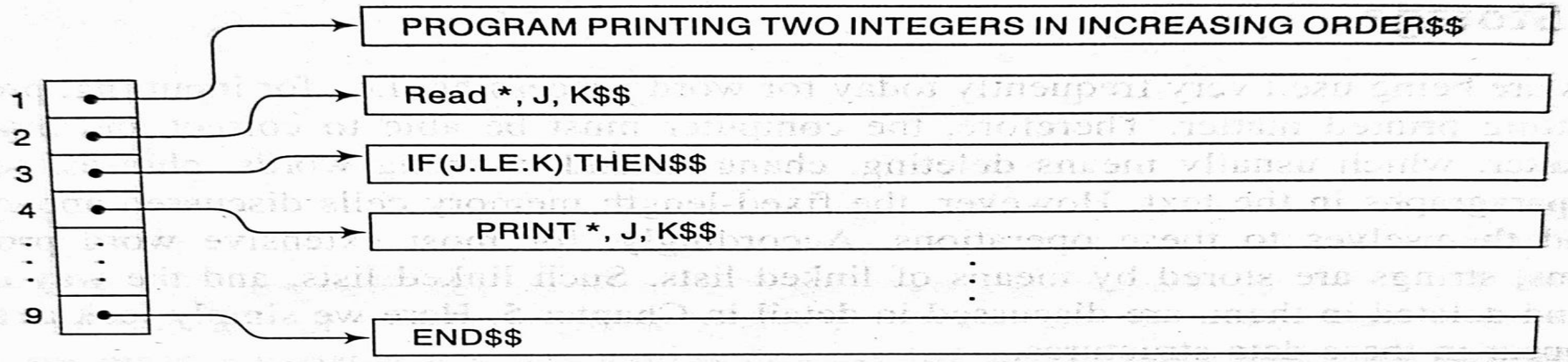


## * Variable-Length storage with Fixed Maximum

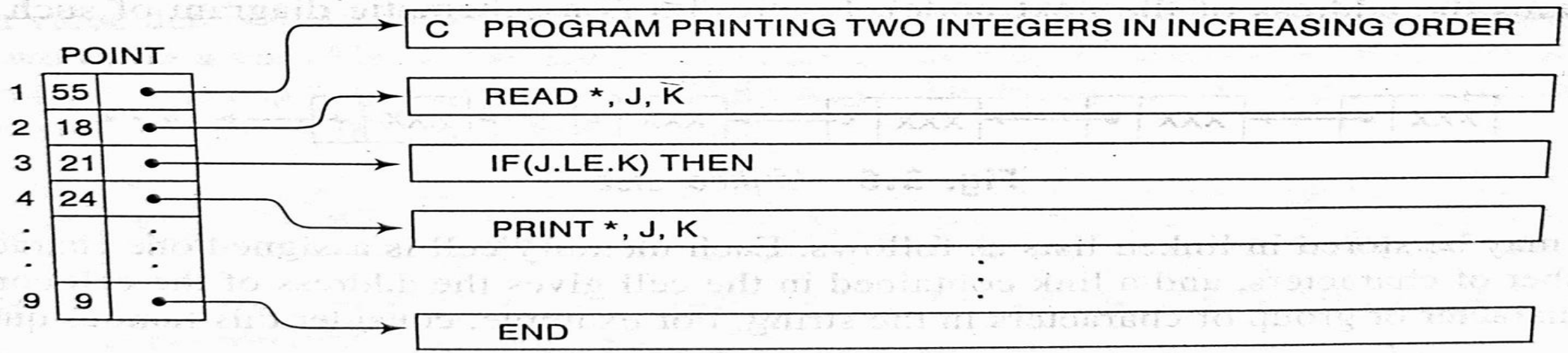
* One then does not have to read the entire record when the string occupies only the beginning part of the memory location.
* Certain string operations depend on having such variable-length strings.
* The storage of variable-length strings in memory cells with fixed lengths can be done in two general ways:

(1) One can use a marker, such as two dollar signs ($$),

(2) One can list the length of the string — as an additional item in the pointer array




(a) Records with sentinels.



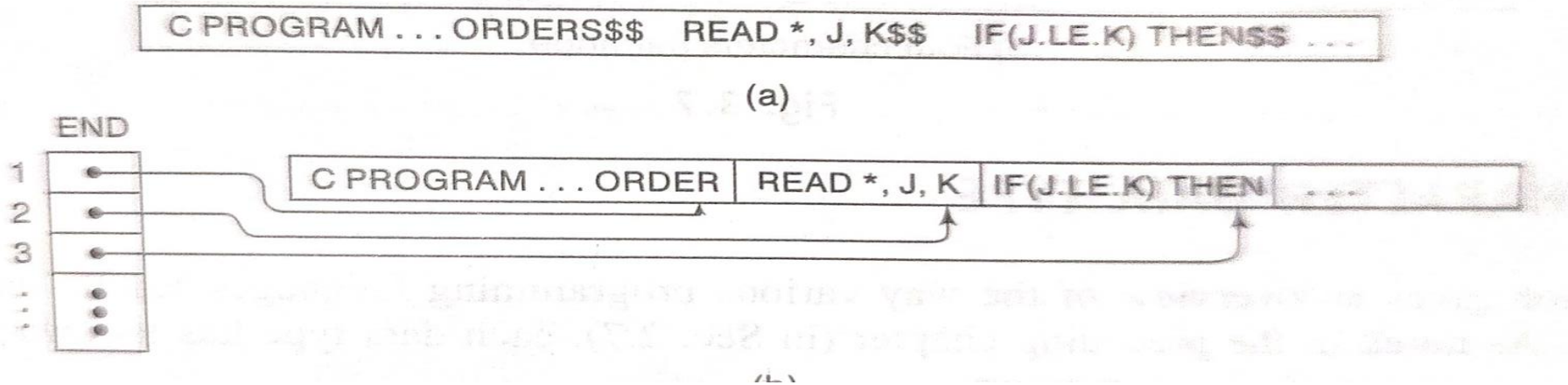

(b) Record whose lengths are listed.

* One might be tempted to store strings one another by using some separation marker, such as two dollar signs ($$)

* Using a pointer array giving the location of the strings.

* This attempts will save space and sometimes used in secondary memory.

* This methods are usually inefficient when the string and their lengths are frequently being changed.

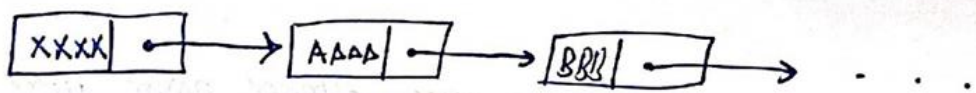


## 3) Linked Storage

→ The fixed length memory cell do not easily lend themselves to : deleting, changing and inserting words, ....

→ For most extensive word processing applications, strings are stored by means of linked lists.

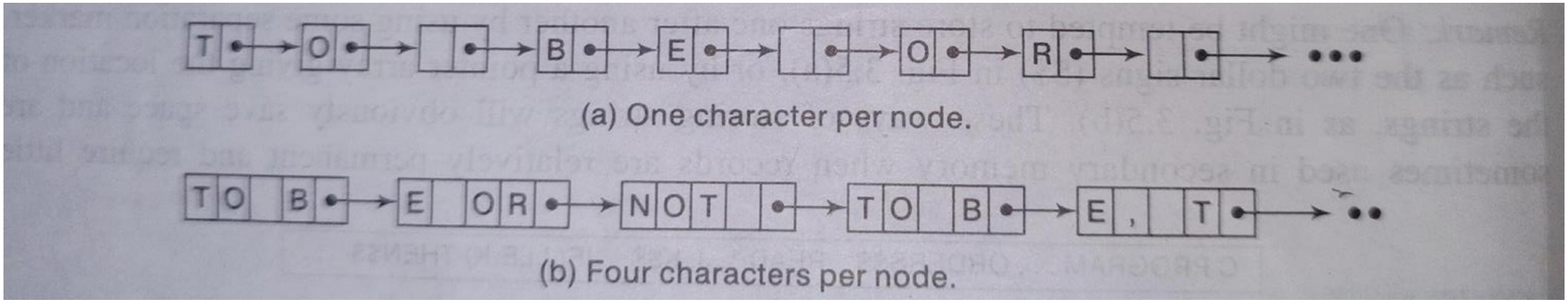
→ Linked list is a linearly ordered sequence of memory cell, called nodes, where each node contains an item called a link, which points to the next node in the list.



Strings may be stored in link lists as follows.

Each memory cell is assigned one character or a fixed number of characters.

The link in the cell gives the cell address containing the next character or group of characters in the string.




(b)

Example

START = 5

| | CHAR | LINK |
|---|---|---|
| 1 | UNIT | 11 |
| 2 | HE P | 8 |
| 3 | | |
| 4 | S | 0 |
| 5 | WE T | 2 |
| 6 | THE | 1 |
| 7 | | |
| 8 | EOPL | 12 |
| 9 | TATE | 4 |
| 10 | | |
| 11 | ED S | 9 |
| 12 | E OF | 6 |

Find the string stored in the Figure.

10 minutes?