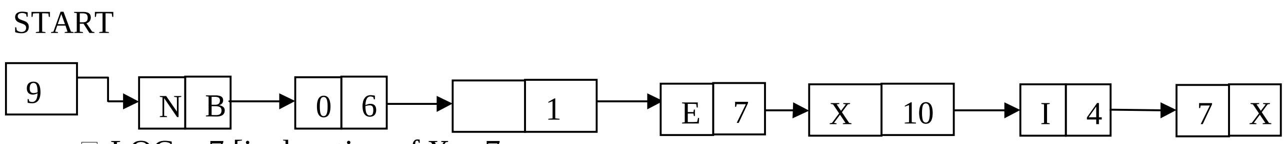
```
ELSE
Set PTR: = LINK [PTR]    [PTR now point to the next node]
[End of li fl structure]
[End of step 2 loop]
(1)  [Search is unsuccessful]
  Set LOC: =NULL
  (5)EXIT
```

Use the first example to show this using the algorithm



 LOC = 7 [i.e location of X = 7

UNIT 5: ARRAY

Linear Array

This is a list of finite number of n of homogeneous data element (i.e data element of the same type) such that:

a) The elements of the array are reference representation by an index set consisting of n-consecutive numbers.

b) The element of the array is stored respectively in successive memory location. Number n of element is called the length or size of the array.

In general, the length or the numbers of the data element can be obtained by the index set of the formular:

Length = UB – LB+1 or length = UB – LB+1

UB = larger index called Upper Bound

LB = smallest index called Lower Bound of the Array.

NB: length = UB when LB=1

The element of an array can be denoted by A_1, A_2, - - - - - - A_n

Example:

Let data is a six element linear array of integer such that:

DATA [1] = 247 DATA [2] = 56 DATA [3] =429

DATA [4] = 135 DATA [5] = 87 DATA [6] =156

DATA 247, 56, 429, 135, 87

This type of array data can be pictured in the form:

DATA

1	247
2	56
3	429
4	135
5	87
6	156

OR

DATA

247	56	429	135	87	156

Example 2:

An automobile company uses an array AUTO to record the number of automobile sold each year from 1932-1984

Solution:

AUTO [K] = number of automobile sold in the years.

Lower Bound = LB = 1932

Upper Bound = UB = 1984

Length = UB – LB+1

= 1984 -1932+1

Length = 53

REPRESENTATION OF LINEAR ARRAY IN THE MEMORY

Let LA be a linear array in the memory of a computer. Recall that the memory of computer is simply a sequence of address location as in figure below;

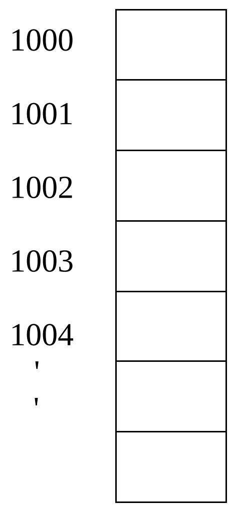
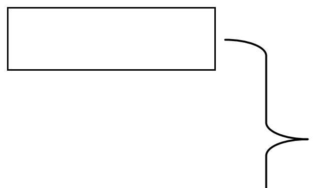


Fig. 1

Let us use the notation:
LOC (LA [K]) =Address of the element LA [K] of the array LA
The computer will not keep track of the entire element but will not only the first element of the list as it will lead it to the other elements
Base (LA) → the first address
LOC (LA [K]) = Base (LA) + w (K-lower bound)
Where w is the words per memory cell of the of the array LA

Example 3:

Consider the array also AUTO in example 2 which record the number of automobile sold each year from 1932 through 1984. Suppose AUTO appear in memory as picture in fig. (2) i.e base AUTO = 200 and w=4 word per memory cell for AUTO.



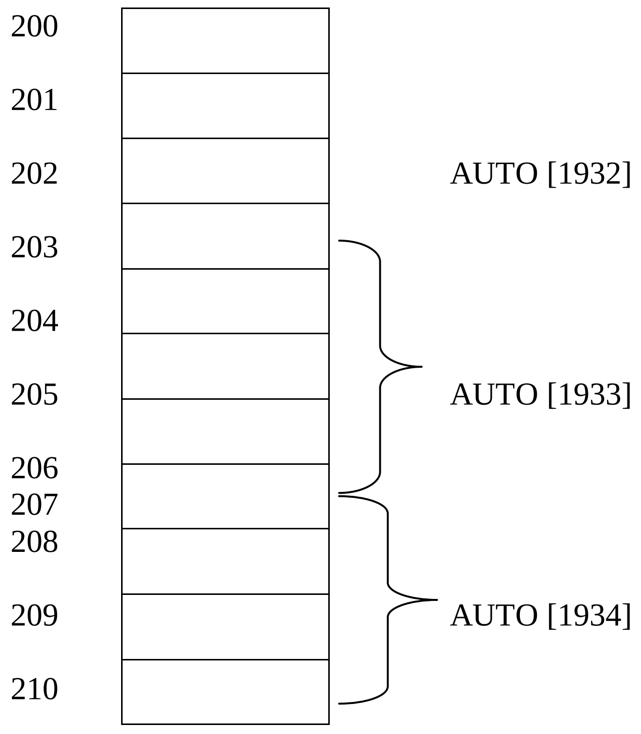


Fig. (2)

LOC (AUTO [1932]) = 200
LOC (AUTO [1933]) = 204
LOC (AUTO [1934]) = 208
The address of the array element for the year K = 1965 can be obtained by using the equation of the formular.
LOC (LA [KK]) = Base (LA) + w(K – Lower bound)
LOC (LA [1965]) = 200+4(1965 – 1932)
=200+4(33) = 200+132 = 332
BASE (LA) = BASE (AUTO) = 200 where w=4, K=1965, LB= 1932
LOC (LA [1965]) = 332.

TRANSVERSING LINEAR ARRAY
Here LA = linear array with lower bound (LB) with upper bound (UB). This algorithm transverse LA applying an operation PROCESS to each element of LA.
ALGORITHM:

```
1.Set K := LB [initialize counter]
2.Repeat step 3 and 4 while K≤UB
3.Apply PROCESS to LA[K] {visit element}
4.Set K: K+1            {increase count}
    [End of step 2 loop]
5.Exit.
```

Algorithm

Transversing a linear Array

```
1.Repeat for K= LB+UB
2.Apply PROCESS to LA[K]
        [End of loop]
3.Exit.
```

Example 4:

Consider example 2, find the number NUM of year during which more than 300 automobile were sold.

Solution: using the algorithm

```
1)      Set NUM := 0 [initialize counter]
2)      Repeat for K = 1932 to 1984
        If Auto [K] □300; then set NUM: = NUM+1
        End of loop
3)      Loop.
```

INSERTING AND DELETING LINEAR ARRAY

Algorithm:

(Inserting into a linear Array) INSERT (LA, N, K, ITEM).

Here LA is a linear array with N elements and K is a positive integer such that $K \leq N$. this algorithm inserts an element ITEM into the Kth position in LA.

```
1.Set J:= N [initialize counter]
2.Repeat for J = K to N- 1
        Set LA [J] = LA [J+1]
         [End of loop]
3.Set N:= N-1
4.Exit.
```

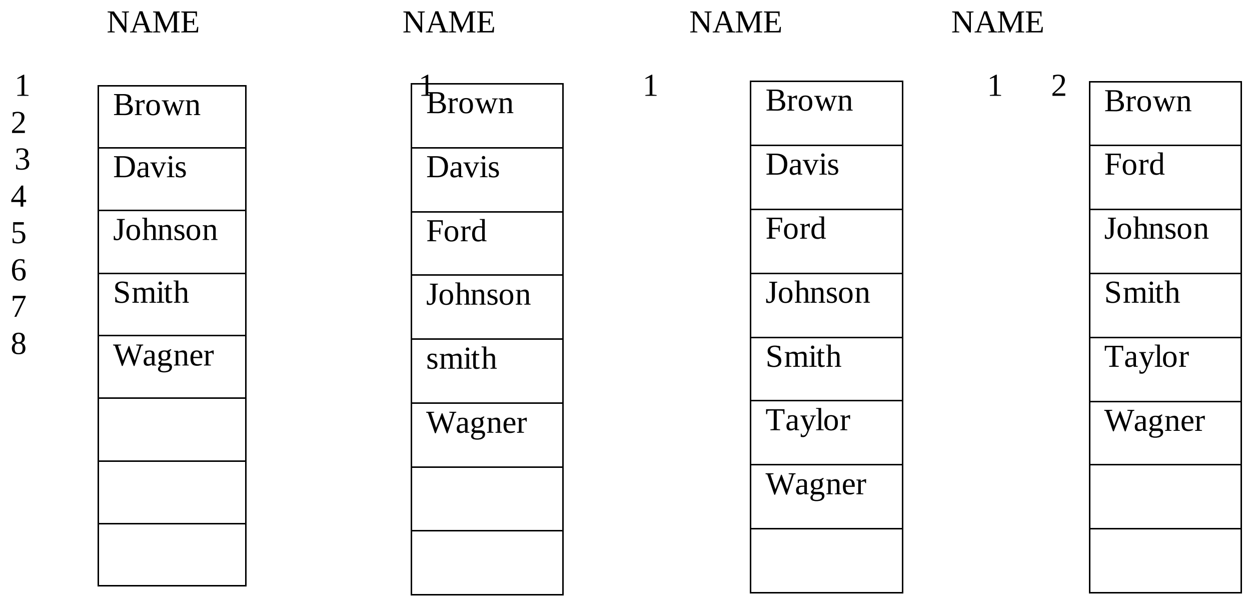
Algorithm:

(Deleting from a Linear Array) DELETE (LA, N, K, ITEM)

Here LA is a Linear Array with N element and K is positive integer such that $K\leq N$. This algorithm deletes the kth element from LA

```
1.Set ITEM := LA[K]
2.Repeat for J = K to N-1
        Set LA [J]:= LA [J+1]
        [End of loop]
3.Set N:= N-1
4.Exit.
```

Example:



MULTIDIMENTIONAL ARRAYS

Two dimensional Array mxn arrays A is a collection of m.n data elements such that each element is specified by a part of integers (such as J, K) called subscripts with the property that $1 \leq J \leq M$ and $1 \leq K \leq n$

The element of A with first subscript J and second subscript K will be denoted by $A_{j.K}$ of A [J, K].

Two dimensional arrays are sometimes called (matrices) matrix array.

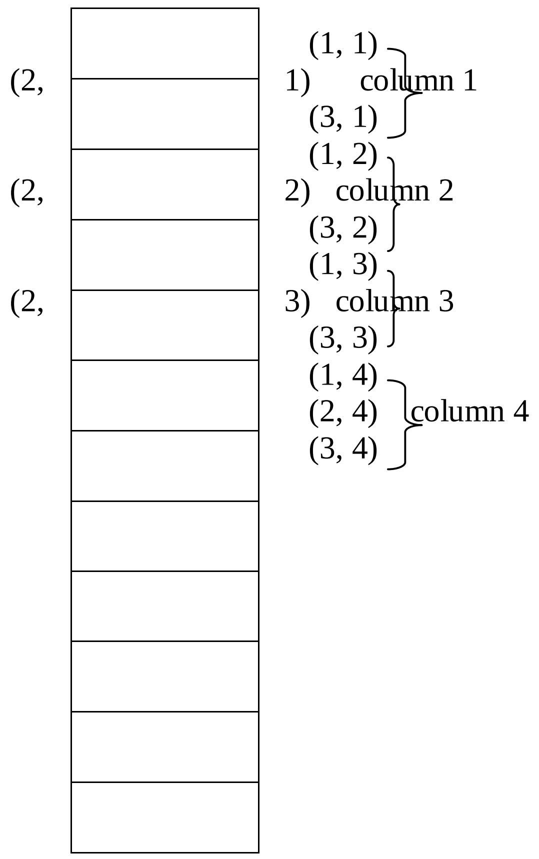
$$\begin{array}{cl} & \text{Column} \\ \text{Row} & \left\{\begin{array}{l} A[1,1], A[1,2], A[1,3], A[1,4] \\ A[2,1], A[2,2], A[2,3], A[2,4] \\ A[3,1], A[3,2], A[3,3], A[3,4] \\ \text{Two dimensional 3X4 Array} \end{array}\right\} \end{array}$$

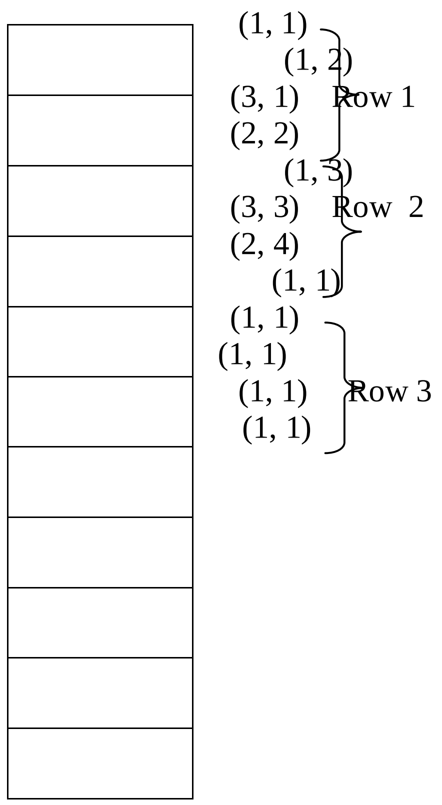
REPRESENTATION OF TWO DIMENSIONAL ARRAYS IN MEMORY

Matrix can be represented in two ways:

1.Column Major Order: 2. Row Major Order sub script:

A subscript


(1, 1)
(2, 1)
(3, 1)
column 1
(1, 2)
(2, 2)
(3, 2)
column 2
(1, 3)
(2, 3)
(3, 3)
column 3
(1, 4)
(2, 4)
(3, 4)
column 4


(1, 1)
(1, 2)
(3, 1)
(2, 2)
Row 1
(1, 3)
(3, 3)
(2, 4)
(1, 1)
Row 2
(1, 1)
(1, 1)
(1, 1)
(1, 1)
Row 3

UNIT 6:   STACKS, QUEUES, RECURSION

A Stack is a linear structure in which items may be added or removed only at one end. Examples of such a structure: a stack of dishes, a stack of pennies and a stack of folded towels. Observe that an item may be added or removed only from the top of any of the stacks.

STACKS

A Stack is an element in which an element may be inserted or deleted only at one end, called the top of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Special terminology is used for two basic operations associated with stacks:

(a) "Push" is the term used to insert an element into a stack.

(b) "Pop" is the term used to delete an element from a stack.

We emphasize that these terms are used only with stacks, not with other data structures.

Examples:

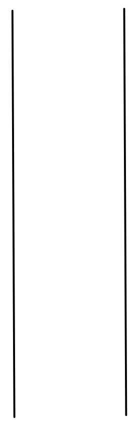
Suppose the following 6 elements are pushed, in order, onto an empty stack:

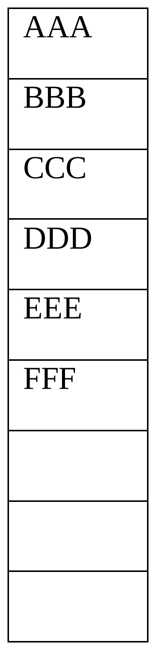
AAA, BBB, CCC, DDD, EEE, FFF

Fig. 2 shows three ways of picturing such a stack. For notational convenience, we will frequently designate the stack by writing:

Stack: AAA, BBB, CCC, DDD, EEE, FFF

The implication is that the right –most elements is the top element. We emphasized that, regardless of the way a stack is described, is underlying property is that insertion and deletion can occur only at the top of the stack. This means EEE cannot be deleted before FFF is deleted, DDD cannot be deleted before EEE and FFF are deleted, and so on. Consequently, the elements may be popped from the stack only in the reverse order of that in which they were pushed onto the stack.





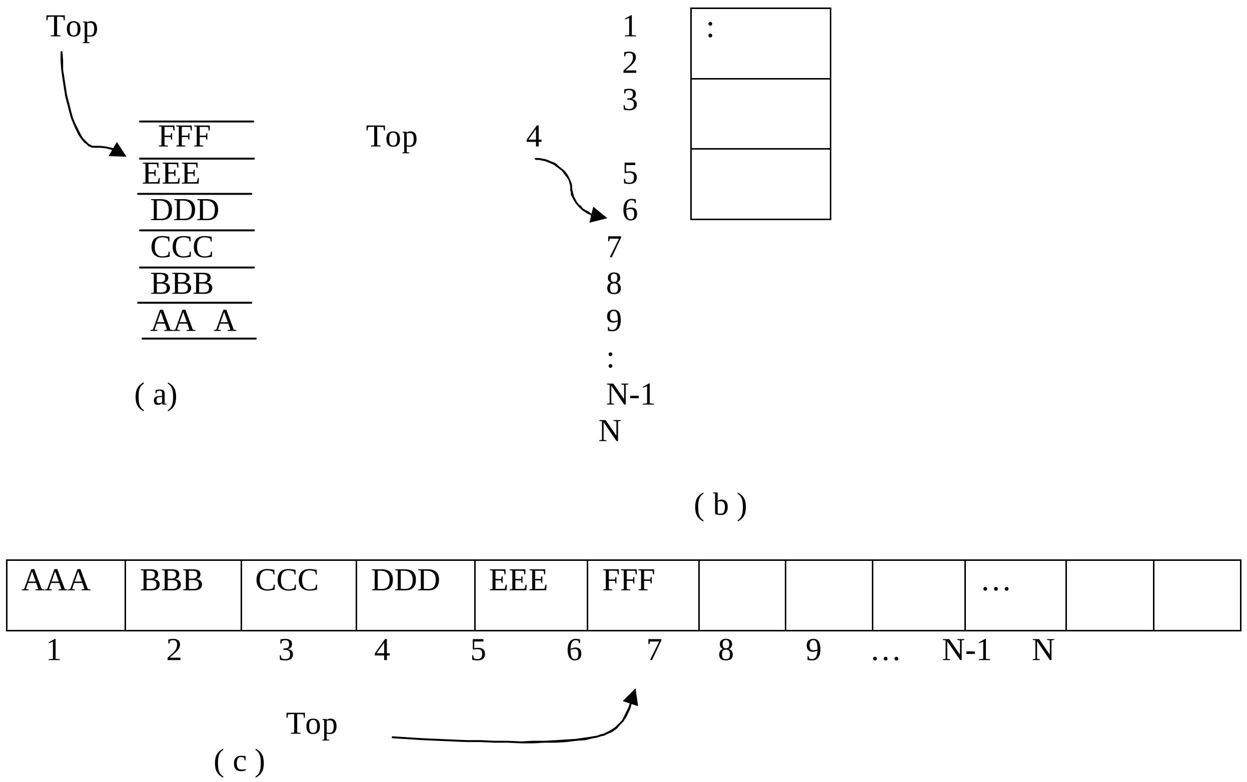


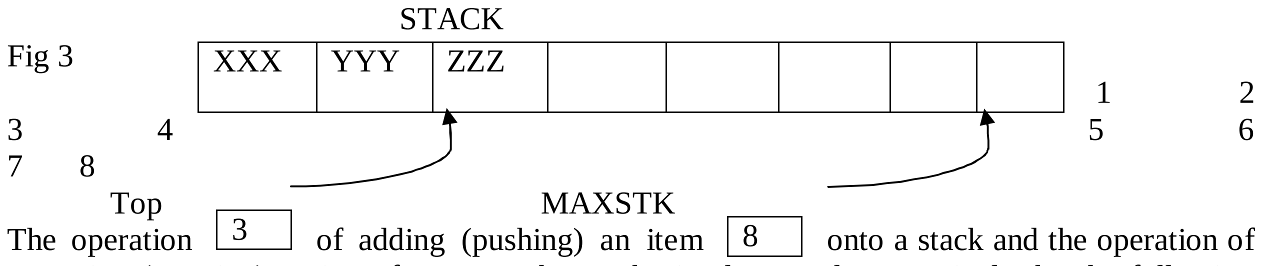
Fig 2 Diagram of stacks

ARRAY REPRESENTATION OF STACKS

Stacks may be represented in the computer in various ways, usually by means of one way list or a linear array. Unless otherwise stated or implied, each of our stacks will be maintained by a linear array STACK; a pointer variable TOP, which contains the location of the top element of the stack; and a variable MAXSTK which gives the maximum number of element that can be held by the stack. The condition TOP =0 or TOP = NULL will indicate that the stack is empty.

Fig 3 pictures such as array representation of a stack (for notation convenience, the array is drown horizontally rather than vertically) since TOP=3, the stack has three element, XXX, YYY, and ZZZ; and since MAXSTK = 8, there is room for 5 more items in the stack.

Fig 3



The operation of adding (pushing) an item onto a stack and the operation of removing (popping) an item from a stack may be implemented, respectively, by the following procedures, called PUSH and POP. In executing the procedure PUSH, one must first test whether there is room in the stack for the new item if not, then we have the condition known as overflow. Analogous, in executing the procedure POP, one must first test whether there is an element in the stack to be deleted; if not, then we have the condition known as underflow.

Procedure: PUSH (STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a stack.

```
1. [Stack already filled?]
   If TOP = MAXSTK, then: print: OVERFLOW, and Return.
2. Set TOP: = TOP + 1. [Increase TOP by 1.]
3.  Set STACK [TOP]:= ITEM. [Inserts ITEM in new TOP position.]
4. Return.
```

Procedure: POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

```
1. [Stack has an item to be removed?]

   If TOP=0, then: print: UNDERFLOW, and return.
2. Set ITEM: = STACK [TOP]. [Assign TOP element to ITEM.]
3. Set: = TOP – 1. [Decrease TOP by 1.]
4.  Return.
```

Frequently, TOP and MAXSTK are global variables; hence the procedures may be called using only

PUSH (STACK, ITEM) and POP (STACK, ITEM)

respectively. We note that the value of TOP is changed before the insertion in PUSH but the value of TOP is changed after the deletion in POP.

ARITHMETIC EXPRESSION; POLISH NOTATION

Let Q be an arithmetic expression involving constants and operations. This section gives an algorithm which finds the value of Q by using reverse Polish (postfix) notation. We will see that the stack is an essential tool in this algorithm.

Recall that the binary operation in Q may have different levels of precedence. Specifically, we assume the following three levels of precedence for the usual five binary operations:

Highest:	Exponentiation (↑)
Next Highest:	Multiplication (*) and division (/)
Lowest:	Addition (+) and subtraction (-)

(Observe that we use the BASIC symbol for exponentiation.) For simplicity, we assume that Q contains no unary operation (e.g., a leading minus sign). We also assume that in any parenthesis-free expression, the operations on the same level are performed from left to right. (This is not standard, since some languages perform exponentiations from right to left.)

Example:

Suppose we want to evaluate the following parenthesis-free arithmetic expression:

$$2 \uparrow 3 + 5 * 2 \uparrow 2 - 12 / 6$$

First we evaluate the exponentiation to obtain

$$8 + 5 * 4 - 12 / 6$$

Then we evaluate the multiplication and division to obtain 8+20-2. Last, we evaluate the addition and subtraction to obtain the final result, 26. Observe that the expression is traversed three times each time corresponding to a level of precedence of the operations.

POLISH NOTATION (PREFIX NOTATION)

For most common arithmetic operations, the operator symbol is placed between its two operands. For example, A + B C – D E * F G / H

This is called *infix notation.* With this notation, we must distinguish between

(A + B) * C and A + (B * C)

By using either parentheses or some operator-precedence convention such as the usual precedence levels discussed above. Accordingly, the order of the operators and operands in an arithmetic expression does not uniquely determine the order in which the operations are to be performed.

Polish notation, named after the polish mathematician Jan Lukasiewiez, refers to the notation in which in which the operator symbol is placed before its two operands. For example,

+AB -CD *EF /GH

We translate, step by step, the following infix expression into polish notation using bracket [] to indicate a partial translation:

(A + B) * C = [+AB] *C = *+ABC

A + (B * C) = A + [*BC] = +A*BC

(A + B) / (C – D) = [+AB] / [-CD] = /+AB – CD

The fundamental property of polish notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operands in the expression. Accordingly, one never needs parentheses when writing expressions in polish notation.

Reverse polish notation refers to the analogous notation in which the operator symbol is placed after its two operands

AB+ CD- EF* GH/

Again, one never needs parentheses to determine the order of the operands in any arithmetic expression written in reverse polish notation. This notation is frequently called *postfix* (or suffix) notation, whereas prefix notation is the term used for polish notation, discussed in the preceding paragraph.