Course Code: **CSC 313**

Course Title: **Data Structure and Algorithms**

Course Unit: **3**

Course Developer/Writer: **A. J. Ikuomola**
&
Dr. A.T. Akinwale

Department of Computer Science
College of Natural Science
University of Agriculture Abeokuta,
Ogun State, Nigeria

UNIT 1: MATHEMATICAL NOTATION AND FUNCTION

Summation Symbol (Sum)

$\sum$ Called Summation (Sigma)

Consider a sequence of a_1, a_2, a_3,… Then the sums

$a_1 + a_2 + a_3 + ... + a_n$ and $a_{m1} + a_{m+1} + \ldots + a_n$

will be denoted respectively by

$$\sum_{j=1}^{n} a_j \quad \text{and} \quad \sum_{j=m}^{n} a_j$$

Example:

(1) $\sum_{i=1}^{n} a_i = a_1 + a_2 + a_3 + a_4 + ... + \ldots + a_n$

(2) $\sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4 + ... + \ldots + a_n b_n$

(3) $\sum_{j=2}^{5} j^2 = 2^2 + 3^2 + 4^2 + 5^2 = 4 + 9 + 16 + 25 = 54$

(4) $\sum_{j=1}^{n} j = 1 + 2 + 3 + 4 + ... + n$

PIE (Product)

$$\pi_{i=1}^{n} x_i = x_1 . x_2 . x_3 ... x_n$$

Floor Function

Let x be any real number, then x lies between two integers called the floor and the ceiling of x. Specifically,

$\lfloor x \rfloor$, called the floor of x denotes greatest integer that does not exceed x.

Examples:

(1) $\lfloor 3.14 \rfloor$ = 3

(2) $\lfloor \sqrt{5} \rfloor$ = 2.23 = 2

(3) $\lfloor -8.5 \rfloor$ = -9

(4) $\lfloor 7 \rfloor$ = 7

Ceiling Function

The symbol for ceiling function is $(\lceil \ \rceil)$ called the ceiling function of x denotes the least integer that is not less than x.

Example:

(1) $\lceil 3.14 \rceil$ = 4

(2) $\sqrt{5}$ = 2.23 = 3

(3) $\lceil -8.5 \rceil$ = - 8

(4) $\lceil 7 \rceil$ = 7

Remainder Function: Modular Arithmetic

Let K be any integer and let M be a positive integer. Then

$$k \pmod{M}$$

(read k modulo M) will denote the integer remainder when k is divided by M. More exactly k (mod M) is the unique integer r such that

$$k = Mq + r \qquad \text{when } 0 < r < M$$

When k is positive, simply divide k by M to obtain the remainder r.

Example:

(1) 25(mod7)

25/7 = 3 r 4

25(mod7) = 4

(2) 25(mod5)

25/5 = 5 r 0

25(mod5) = 0

(3) 35(mod11)

35/11 = 3 r 2

35(mod11) = 2

(4) 3(mod8)

3/8 = 3 r 4

3(mod8) = 3 (note that 3 = 8 . 0 + 3 = 3) when q= 0

UNIT 2: DATA STRUCTURE

Introduction

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structure is the logical arrangement of data element with the set of operation that is needed to access the element. The logical model or mathematical model of the particular organization of data is called a data structure. It is defined as a set of rules and constraint which shows the relationship that exist between individual pieces of data which may occur.

Basic Principle

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address – a bit string that can be stored in memory and manipulated by the program. Thus the record and array data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing addresses of data items within the structure itself. Many data structures use both principles.

The choice of a data structure for a particular problem depends on the following factors:

1) Volume of data involved
2) Frequency and ways in which data will be used.
3) Dynamic and static nature of the data.
4) Amount of storage required by the data structure.
5) Time to retrieve an element.
6) Ease of programming.

Classification of Data Structure

(1) Primitive and non – primitive: primitive data structures are basic data structure and are directly operated upon machine instructions. Examples are integer and character. Non-primitive data structures are derived data structure from the primitive data structures. Examples are structure, union and array.

(2) Homogenous and Heterogeneous: In homogenous data structures all the elements will be of the same type. Example is array. In heterogeneous data structure the elements are of different types. Example: structure

(3) Static and Dynamic data structure: In some data structures memory is allocated at the time of compilation such data structures are known as static data structures. If the allocation of memory is at run-time then such data structures are known as Dynamic data structures. Functions such as malloc, calloc, etc. are used for run-time memory allocation.

(4) Linear and Non – linear data structure: Linear data structure maintains a linear relationship between its elements. A data structure is said to be linear if its elements form a sequence or a linear list. Example, array. A non-linear data structure does not maintain any linear relationship between the elements. Example: tree.

Linear structure can be represented in a memory in 2 basic ways:

i) To have the linear relationship between the element represented by mean of sequential memory location. These linear structures are called ARRAY.
ii) To have the linear relationship between the elements represented by means of points or links. These linear structures are called LINKLIST.

Data Structure Operation

The following operations are normally performs on any linear structure, whether is an array or a linked list.

- Transversal (Traversing)
- Search (Searching)
- Inserting
- Deleting
- Sorting
- Merging

Transversal/Transversing: accessing each element or record in the list exactly only, so that certain items in the record may be processed. This accessing and processing is sometimes called “visiting” the record.

Search (Searching): finding the location of the record with a given key value or finding the location of all records which satisfy one or more conditions.

Inserting: adding a new record to the structure

Deleting: removing an element from the list of records from the structure.

Sorting: arranging the record in some logical order (e.g. alphabetically according to some NAME key or in numerical order according to some NUMBER key such as social security number, account number, matric number, etc.)

Merging: combining the records in two different sorted file into a single sorted file.

Characteristics of Data Structures

Data Structure	Advantages	Disadvantages
Array	Quick inserts Fast access if index know	Slow search Slow deletes Fixed size
Ordered Array	Faster search than unsorted array	Slow inserts Slow deletes Fixed size
Stack	Last-in, first-out access	Slow access to other items
Queue	First-in, first-out access	Slow access to other items
Linked List	Quick inserts Quick deletes	Slow search
Binary Tree	Quick search Quick inserts Quick deletes (if the tree remains balanced)	Deletion algorithm is complex
Red-Black Tree	Quick search Quick inserts	Complex to implement

	Quick deletes (Tree always remains balanced)	
2-3-4 Tree	Quick search Quick inserts Quick deletes (Tree always remains balanced) (Similar trees good for disk storage)	Complex to implement
Hash Table	Very fast access if key is known Quick inserts	Slow deletes Access slow if key is not known Inefficient memory usage
Heap	Quick inserts Quick deletes Access to largest item	Slow access to other items
Graph	Best models real-world situations	Some algorithms are slow and very complex

UNIT 3: HASH FUNCTION

A hash function is any well defined procedure or mathematical function that converts a large, possibly variably sized amout of data into small datum usually a single integer that may serve as an index to an array. The value returns by hash function are called Hash value, Hash Codes, Hash sums or simply Hashes.

The function $H = K \rightarrow L$ is called a hash function.

The two principal criteria used for selecting a hash functions $H = K \rightarrow L$ are as follows:

(1) The hash function H should be very easy and quick to compute.

(2) The function H should as far as possible, uniformly distribute the hash addresses throughout the set L so that there are minimum number of collision.

Hash Function Techniques

1. Division method
2. Mid-square method
3. Folding method

1. Division Method

Choose a number m larger than the number n of keys in K (the number m is usually chosen to be a prime number or a number without small division, since these frequently minimizes the number of collision). Then the hash function H is denoted by;

$$H(k) = k \pmod{m} \text{ or } H(k) = k \pmod{m} + 1$$

The first formula k (mod m) denotes the remainder when k is divided by m while the second formular is used when we want the hash address to range from 1 to m rather than from 0 to m-1.

Example:

Suppose a company with 68 employees assign a 4 - digit employee number to each employee which is used as the primary key in the company's employee file. Suppose L consist of 100 two-digit addresses 00, 01, 02, …, 99. Applying the hash function to each of the following employee numbers: 3205, 7148, 2345.

Solution:

Using division method, choose a prime number in which is close to 99 such as m = 97. Then

H(k) = k (mod m)

a) H(3205) = 3205(mod97) = 3205/97 = 4 H(3205) = 4

That is, dividing 3205 by 97 gives a remainder of 4.

b) H(k) = k (mod m)

H(7148) = 7148(mod97) = 7148/97 = 67 H(7148) = 64

That is, dividing 7148 by 97 gives a remainder of 64.

c) H(k) = k (mod m)

H(2345) = 2345(mod97) = 2345/97 = 17 H(2345) = 17

That is, dividing 2345 by 97 gives a remainder of 17.

2. Midsquare

The key k is square. Then the hash function H is defined by

$$H(k) = l$$

where l is obtained by deleting digits from both ends of k^2. Note that the same position of k^2 must be used for all of the keys.

Example

Using the above equation

Solution

The following calculations are performed:

k	3205	7148	2345
K^2	10272025	51093904	05499025
H(k)	72	93	99

Observe that the fourth and fifth digits, counting from the right, are chosen for the hash address .

3. Folding Method

The key k is partitioned into a number of parts $k_{1, \ldots,} k_{r,}$ where each parts, except possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry. That is,

$$H(k) = k_1 + k_2 + k_3 + \ldots + k_r$$

where the leading-digit carries, if any, are ignored. Sometimes, for extra "milling", the even-numbered parts k_2, $k_{4, \ldots,}$ are each reversed before the addition.

Example:

Chopping the key k into two parts and adding yields of the following hash addresses:

H(3205) = 32 + 05 = 37

H(7148) = 71 + 48 = 119 = 19

H(2345) = 23 + 45 = 68

Observe that the the leading digit **1** of H(7148) is ignored. Alternatively, one may want to reverse the second parts before adding, this producing the following hash addresses:

H(3205) = 32 + 50 = 82

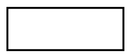
H(7148) = 71 + 84 = 155 = 55

H(2345) = 23 + 54 = 77

UNIT 4: LINKED LIST

Basic Concepts

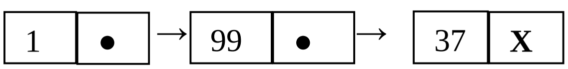
This is a data structure that consist of a sequence of data record such that in each record there is a field that contain a reference to the next field


→ [Info link]

A node is made up of two parts which are the data field and link-list.

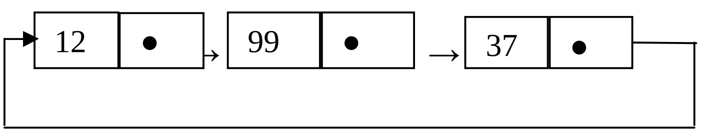
Each record of a link-list is called a NODE which is made up of two parts the information part and the pointer part.

Linear List



In linear linked list, the components are all linked together in some sequential manner.

Circular List



In circular linked list, the component has no beginning and end.

Singly, doubly and multiply linked list are example of a linked list:

Singly-linked list contain nodes which have a data field as well as a next field, which points to the next node in the linked list.

In a doubly-linked list, each node contains, besides the next-node link, a second link filed pointing to the previous node in the sequence. The two links may be called forwars(s) and backwards.

A Multiply Linked List

Representation of Link List in Memory

Let LIST be a linked list. LIST require two linear arrays called INFO and LINK, such that INFO [K] and LINK [K]contain, respectively, the information part and the next pointer field of a node of LIST. It should be noted that, LIST requires a variable name such as START which indicate the beginning of the list and a next-pointer sentinel – denoted by NULL which indicate the end of the list.

Example

START: 9

	INFO	LINK
1		
2		
3	O	6
4	T	0
5		
6	□	11
7	X	10
8		
9	N	3
10	I	4
11	E	7
12		

Interpreted as:

START = 9, so INFO [9] = N (is the first character)

LINK [9] = 3, so INFO [3] = O (is the second character)

LINK [3] = 6, so INFO [6] = □ (blank) is the third character

LINK [6] =11, so INFO [11] = E is the fourth character

LINK [11] = 7, so INFO [7] = X is the fifth character

LINK [7] = 10, so INFO [10] = I is the sixth character

LINK [10] = 4, so INFO [4] = T is the seventh character

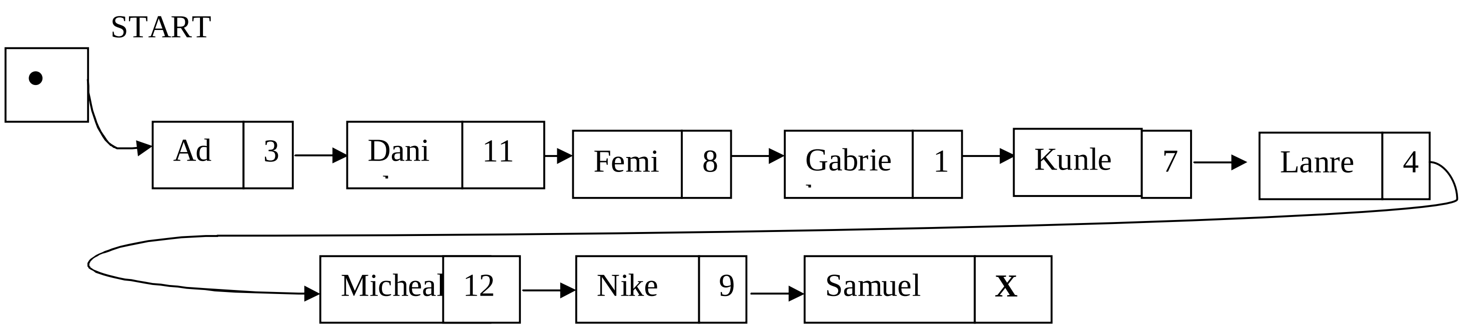
LINK [4] = 0 INFO [0] = NULL value, so the list has ended

In other words, NO EXIT is the character string

Example:
A hospital ward contains 12 beds of which 9 are occupied. The listing is given by pointer field START

5

Bed Number	Patient	
1	Kunle	7
2		
3	Daniel	11
4	Micheal	12
5	Ade	3
6		
7	Lanre	4
8	Gabriel	1
9	Samuel	0
10		
11	Femi	8
12	Nike	9



Example 2:

This figure shows the test score in Algebra & geometry stored in the same linked list.

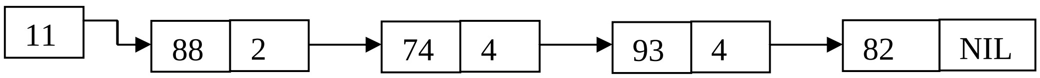
ALG: 11

GEOm: 5

	TEST	LINK
1		
2	74	14
3		
4	82	0
5	84	12
6	78	0
7	74	8
8	100	13
9		
10		
11	88	2
12	62	7
13	74	6
14	93	4
15		
16		

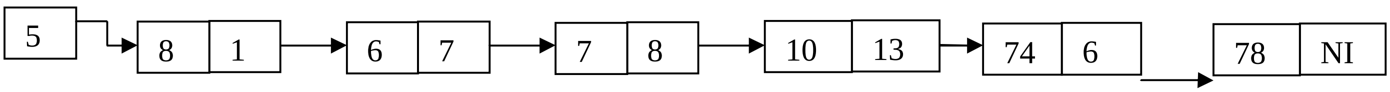
FOR ALGEBRA (ALG)

ALG



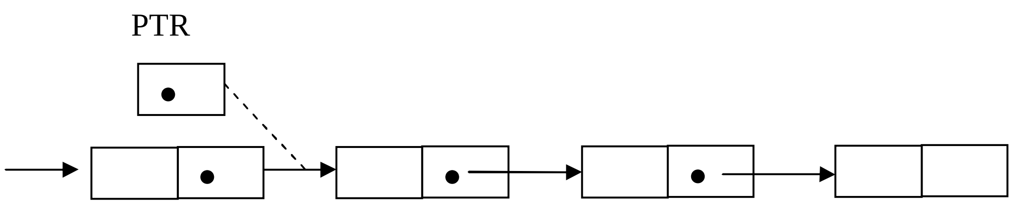
The information for ALG is 88, 74, 93, 82

FOR GEOM



The information for geom. is 84, 62, 74, 100, 74, and 78

Transversing a Link List



PTR = LINK [PTR]

Algorithm to access each element exactly once in the list

(1) Set PTR = START [initiate pointer PTR]

(2) Repeat step 3 and 4 while PTR = NULL

(3) Apply process to INFO [PTR]

(4) Set PTR : = LINK [PTR] [PTR now points to the next node] [End of step 2 loop]

(5) Exit

Searching

Algorithm 2:

List is a linked list in memory. This algorithm finds the location LOC of node where ITEM first appear in LIST or sets LOC = NULL

(1) Set PTR : = START

(2) Repeat step 3 while PTR NULL

(3) If ITEM = INFO [PTR] , then

Set LOC: = PTR and EXIT

ELSE

Set PTR: = LINK [PTR]   [PTR now point to the next node]

[End of li fl structure]

[End of step 2 loop]

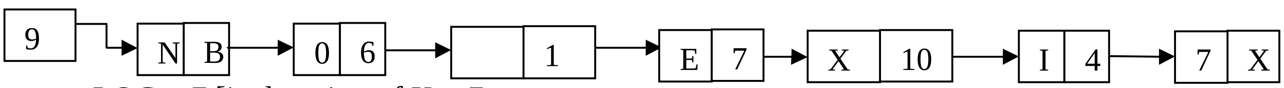
(4) [Search is unsuccessful]

Set LOC: =NULL

(5)EXIT

Use the first example to show this using the algorithm

START



∴ 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;

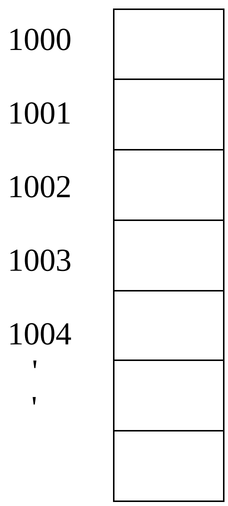


Fig. 1

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

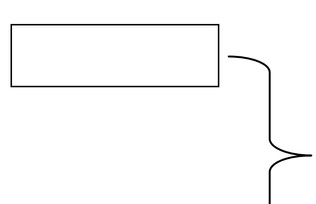
Base (LA) → the first address

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.



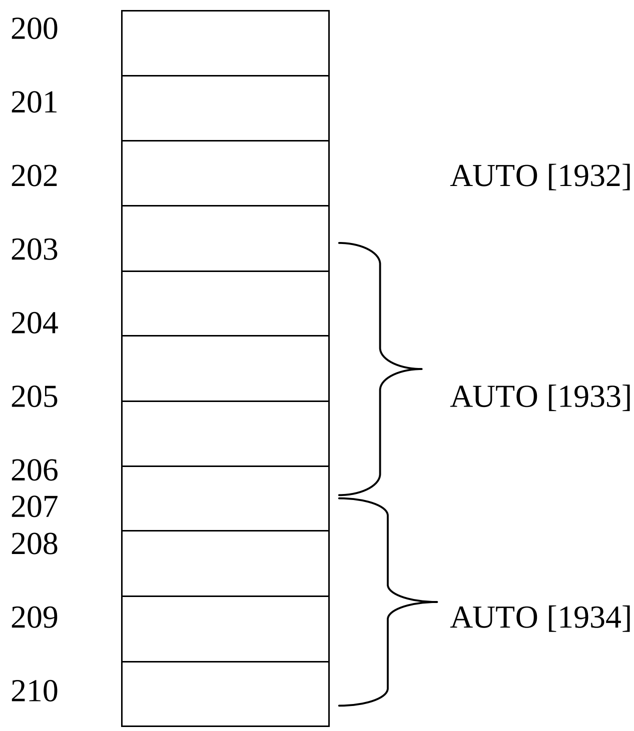


Fig. (2)

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 ≤N. this algorithm inserts an element ITEM into the K^{th} 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≤ N. This algorithm deletes the k^{th} 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:

	NAME	NAME	NAME	NAME
1	Brown	1 Brown	1 Brown	1 2 Brown
2	Davis	Davis	Davis	Ford
3	Johnson	Ford	Ford	Johnson
4	Smith	Johnson	Johnson	Smith
5	Wagner	smith	Smith	Taylor
6		Wagner	Taylor	Wagner
7			Wagner	
8				

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≤ J≤ M and 1≤ K≤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.

Column

Row { 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] }

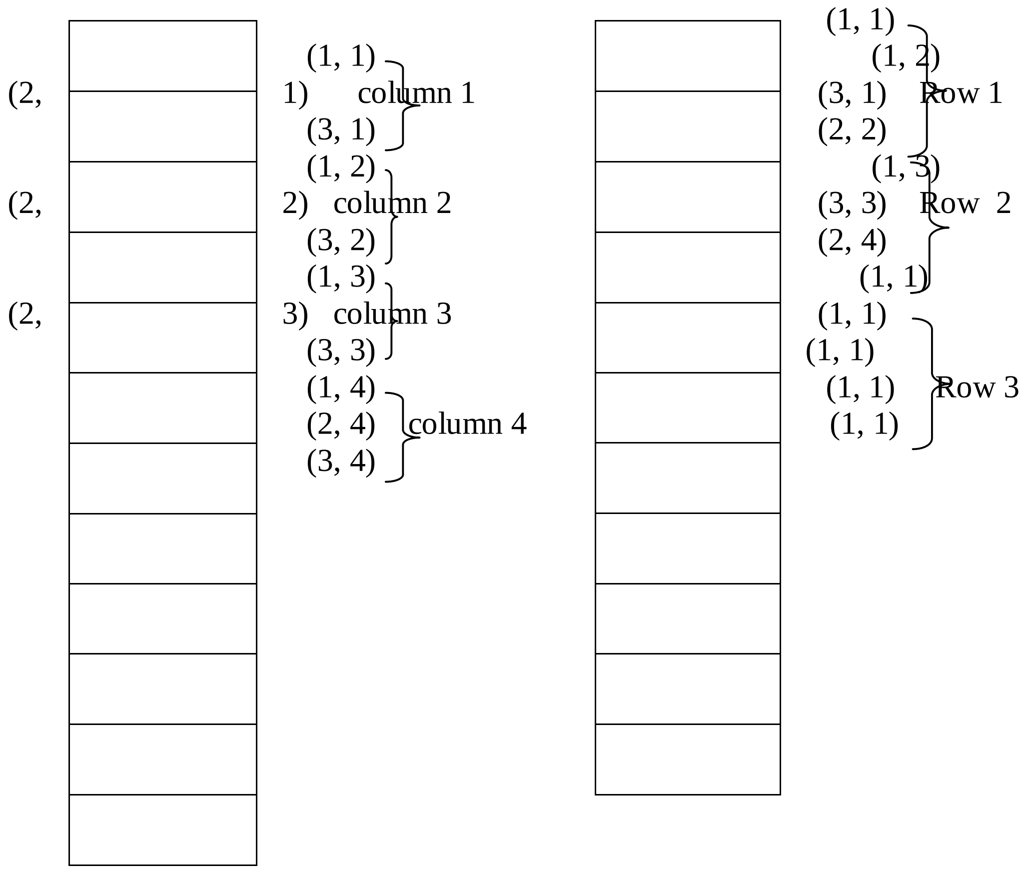
Two dimensional 3X4 Array

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:

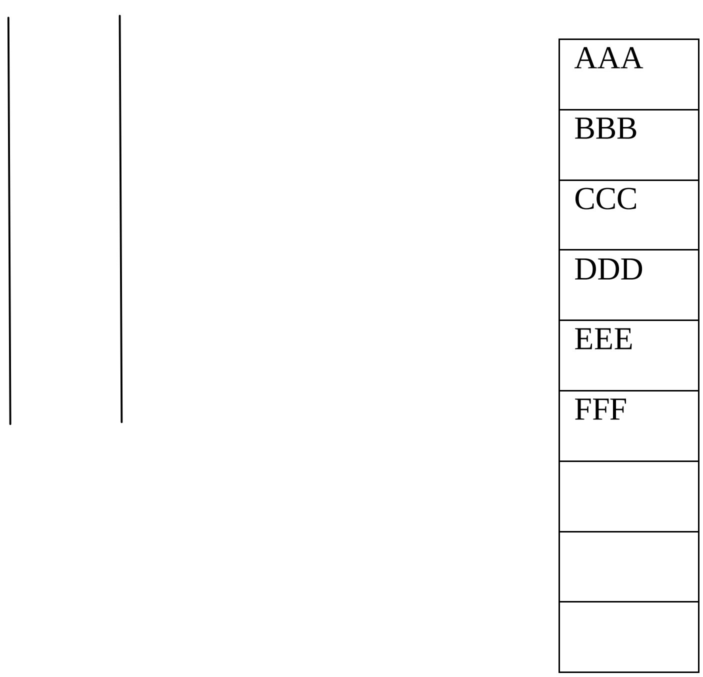
Suppose the following 6 elements are pushed, in order, onto an empty stack:

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.



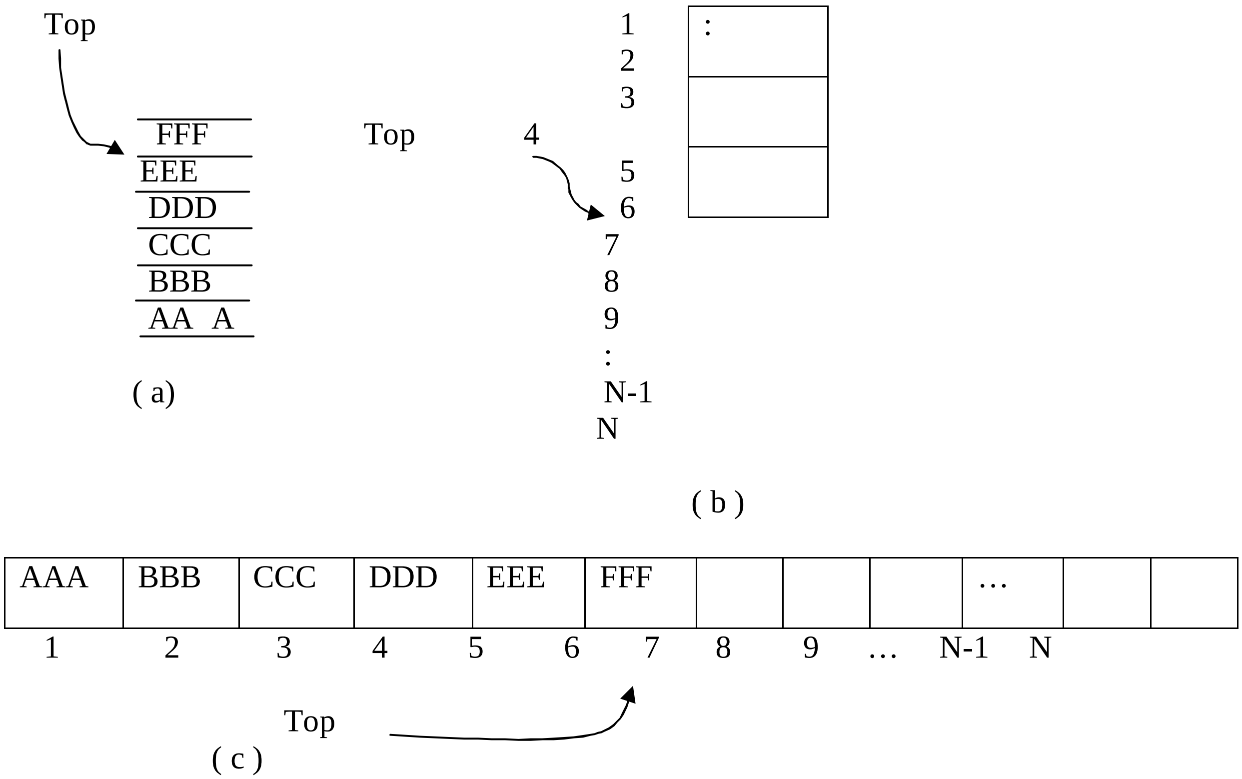


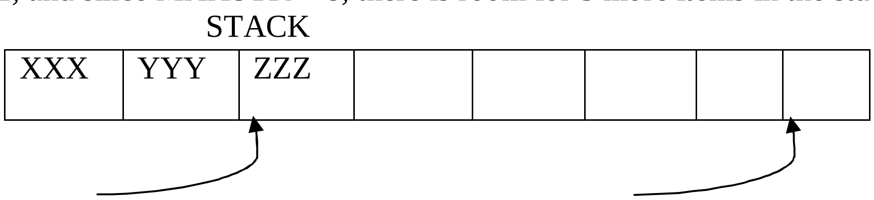
Fig 2 Diagram of stacks

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



1 2
3 4 5 6
7 8

Top 3 MAXSTK 8

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.