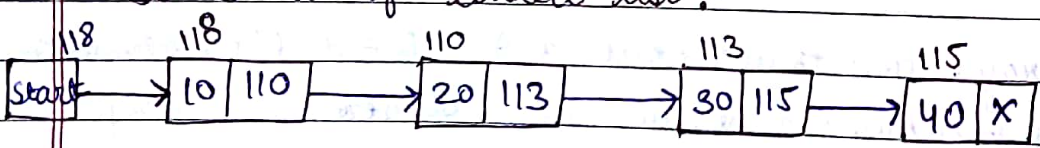
Unit 2

Difference between Array and linked list :-

Array	Linked List
1. Array is a collection of element having same data type with common name.	1. Linked List is an ordered collection of elements which are connected by links / pointers.
2. In array, elements can be accessed using index / sub--script value, i.e. elements can be randomly accessed like arr[0], arr[1] etc. So array provides fast and Random access	2. In linked list, element can't be accessed randomly but can be accessed only sequentially and accessing element takes O(n) time.
3. In array, elements are stored in consecutive manner in memory.	3. In linked list, elements can be stored at any available place as address of node is stored in previous node.
4. Insertion & deletion takes more time in array as elements are stored in consecutive memory locations.	4. Insertion & deletion are fast and easy in linked list as only value of pointer is needed to change.

Array	Linked List
5. Array can be single dimensional, two dimensional or multidimensional.	5. Linked list can be singly doubly or circular linked list.
6. In array, each element is dependent, no connection with previous element or with its location.	6. In linked list, location or address of element is stored in the link part of previous element / node.
7. In array, no pointer are used link linked list so no need of extra space in memory for pointer.	7. In linked list, adjacency between the elements are maintained using pointers or links, so pointers are used and for that extra memory space is needed.

structure of linked list :-




```
struct node
{
    int info;
    struct node *link;
};
    struct node * start;
```

Linked List:- Linked list is a collection of nodes There are two parents in single node. First part contain information and second part contain link of the next node. In last node link part ~~convert NUL~~ contain NULL Value and first node contain START.

struct node * P = (struct node *) malloc (size of (struct node));

Operations

1. Traversing in a linked list :-

Algo

1. Set Ptr := start;
2. Repeat step 3 & 4 while Ptr != NULL do
3. Apply PROCESS to Info [Ptr]
4. Set Ptr := link [Ptr].
5. Exit

2. Counting in a linked list :-

Algo.

1. Set Ptr := start and count := 0.
2. Repeat step 3 to 4 while Ptr != NULL do ~~and Info[Ptr] != Ele.~~
3. count := count + 1
4. Set Ptr := link [Ptr]
5. Print count
6. Exit.

3. Searching in a linked list :-

Algo

1. Set Ptr := start
2. Repeat 3 while Ptr != NULL do and ~~Info~~
3. If info [Ptr] = ELE
   Set LOC := Ptr and go to step 5.

else
set Ptr := link [Ptr]

4. Set LOC := NULL
5. Print LOC
6. Exit.

4. Searching in sorted linked list
Algo
1. set Ptr := start
2. Repeat step 3 while Ptr != NULL
3. If Ele > Info [Ptr] then
set ptr := link [Ptr]
else
if ele = info [Ptr] then
set LOC := ptr and exit
else
LOC := NULL and Exit
4. Set LOC := NULL
5. Exit.

5. Insertion in linked list :-

(i) at the begining of linked list:-
Algo.
1. If Avail = NULL then
write: Memory overflow and Exit
2. set new := Avail and Avail := link [Avail]
3. Set Info [new] := Ele
4. link [new] := Start
5. start := new
6. Exit.

Scanned by CamScanner

(ii) at the end of the Linked list:-

Algo

1. If Avail = NULL then
   write: memory overflow and exit
2. set new := Avail and Avail := link [Avail]
3. set Info [New] := Ele and link [new] := NULL
4. Set P := start
5. Repeat while link [P] != NULL
   P := link [P]
6. set link [P] := new
7. Exit.

(iii) after a given node in linked list:-

Algo

1. If Avail = NULL then
   write: memory overflow and exit
2. Set new := Avail and Avail := link [Avail]
3. set Info [New] := Ele and link [new] := NULL
4. Set P = start
5. Repeat while (P != NULL and Info [P] != X]
   P := link [P]
6. If P = NULL then
   write element not found
   else
   (a) link [new] := link [P]
   (b) link [P] := new
7. Exit.

(iv) Before a given node in linked list :-

Algo

1. If Avail := NULL then
   write: memory overflow and exit
2. Set New := Avail and Avail := Link [Avail]
3. Set Info [New] := Ele and link [new] := NULL
4. Set Ptr := start and temp := NULL
5. Repeat while (Ptr != NULL and Info [Ptr] != X)
   {
   Set temp := ptr
   Set ptr := link [Ptr]
   }
6. If Ptr = NULL then
   write element not found
   else
   (a) link [new] := ptr
   (b) link [temp] := new
7. Exit.

—x—

(v) In the sorted linked list :-

Algo

1. If Avail := NULL then
   write: memory overflow and exit
2. Set New := Avail and Avail := link [Avail]
3. Set Info [New] := Ele and link [New] := NULL
4. Set temp := NULL and Ptr := start
5. Repeat while (Ptr != NULL and Info [Ptr] < ele)
   {
   temp := ptr
   Ptr := link [Ptr]
   }

6. If Ptr = NULL
   link [temp] := new
   else if temp == NULL
   start := new
   else
   link [new] := ptr
   link [temp] := new
7. Exit.

6. Deletion in linked list:-

(i) at the beginning of linked list:-
Algo
1. If start = NULL then
   write: underflow and exit
2. set temp := start
3. start := link [start]
4. link [temp] := Avail and Avail := temp
5. Exit.

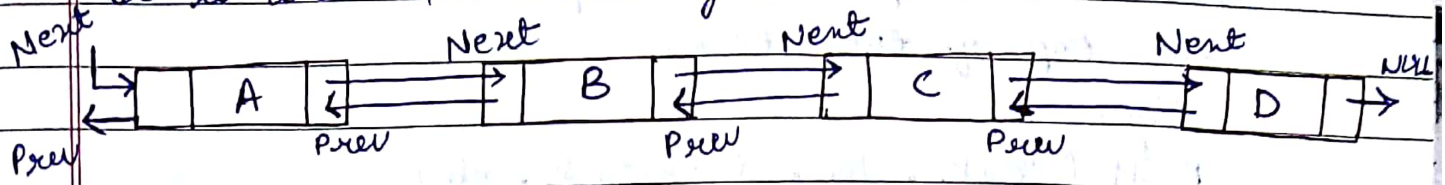
(ii) at the ending of linked list:-
Algo
1. If start = NULL then
   write: underflow and exit
2. Set temp := start and Pre := NULL
3. Repeat while link [temp] != NULL
   {
   pre := temp
   temp := link [temp]
   }

4. If (pre = NULL)
   start := NULL
   Else
   link [Pre] := NULL
5. link [temp] := Avail and Avail := temp
6. Exit.

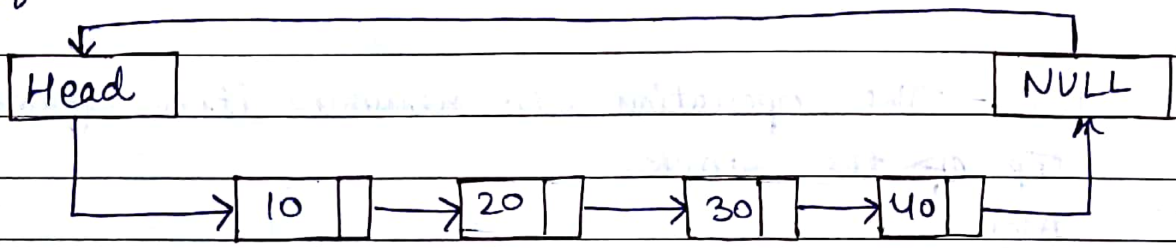
(iii) A given node in linked list :-
Algo
1. If start = NULL then
   write: underflow and exit
2. Set temp := start and Pre := NULL
3. Repeat while temp != NULL and Info [temp] != Ele
   {
   pre := temp
   temp := link [temp]
   }
4. If temp = NULL then
   write: "Element not found" and exit.
   else
   If pre = NULL then
   start := link [start]
   else
   link [Pre] := link [temp].
5. free (temp)
6. Exit

Doubly Link list :-
It contains an extra pointer, typically called previous pointer together with the next pointer and data which are there in singly linked list.

It is also k/a two way list.



Circular Linked List :-
Circular list is a list in which the link field of the last node is made to point to the start / first node of the list.



Doubly circular Linked list :-
- In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points of the last node making the circular in both direction.
- Insertion and deletion at specified position is same as doubly link list.

—X—

STACK :-
- Stack is a collection of elements, where element last to be inserted is first to be taken out.
- Stack implement principle of FILO or LIFO.
- Only access to the stack is the top element.

Operations on stack :-
1. PUSH
2. POP

1. PUSH :- The operation to place a new item at the top of the stack.

Algo.

PUSH (stack, Top, Maxstack, Ele)

1. If TOP = MaxStack then
   write: stack overflow and exit.
2. Set TOP := TOP+1
3. Stack [TOP] := Ele.
4. Return.

2. POP :- The operation to remove item from the top of the stack.

Algo.

POP(stack, top, Ele)

1. If TOP = 0 then
   write: stack underflow and exit
2. Set Ele := stack [TOP]
3. Set TOP = TOP-1
4. Return.

Application of Stack :-

1. Function calling / Recursion
2. Matching Parenthesis
3. Compilation of Arithmatic Expressions
4. Convert no. bases.
5. Depth first search.

Arithmetic expressions:-

1. Infix expression (notation)
2. Prefix expression (Polish notation)
3. Postfix expression (Reverse Polish notation).

Evaluation of postfix expression:-

Suppose P is an arithmatic expression written in postfix notation, the following algorithm which usases a STACK to hold operands evaluats P.

Algorithm:- This algorithm finds the value of an arithmatic expression and written in postfix notation.

1. Add a right parenthesis ")" at the end of P.

2. Scan P from left to right and repeat steps 3 and 4 for each element of P unit ")" is encountered.

3. If an operand is encountered then put it on stack.

4. If an operator is encountered then
(a) Remove the two top element of stack where A is the top and B is the next of top element.

(b) evaluate B⊗A

(c) Place the result B back on the STACK.

5. Set value := top element of STACK

6. Exit.

Conversion of Infix to Postfix :- (Q_1, P_1)

Algorithm:-

1. PUSH "(" onto stack and add ")" to the end of Q.

2. Scan Q_1 from left to right and repeat step 3 to 6 for each element of Q_1 until stack is empty.

3. If an operand is encountered add it to P_1.

4. If a "(" paranthesis is encountered then PUSH it in the STACK.

5. If an operator ⊗ is encountered then:-

(a) Repeatly pop from stack and add to P_1 each operator (on the top of stack) which has the same procedure as or higher precedence than ⊗.

(b) Add ⊗ to STACK.

6. If a ")" paranthesis is encountered then:-

(a) Repeatly pop from stack and add a P_1 each operator (to the top of stack) until a left parenthesis is encountered.

(b) Remove the left parenthesis.

7. Exit.

Recursion :-

Recursion have 2 properties:-

1. These must be certain criteria called base criteria for which recursion doesn't call itself.

2. Each time the procedures does call itself it must be closer to base criteria.

factorial function :-

If $n = 1$, then $n! = 1$

If $n > 1$, then $n! = n * (n-1)!$

FACTORIAL (Fact, n)

1. If N = 1 then
   Set fact := 1 and return

2. call FACTORIAL (fact, n-1)

3. Set Fact := n * FACT

4. Return.