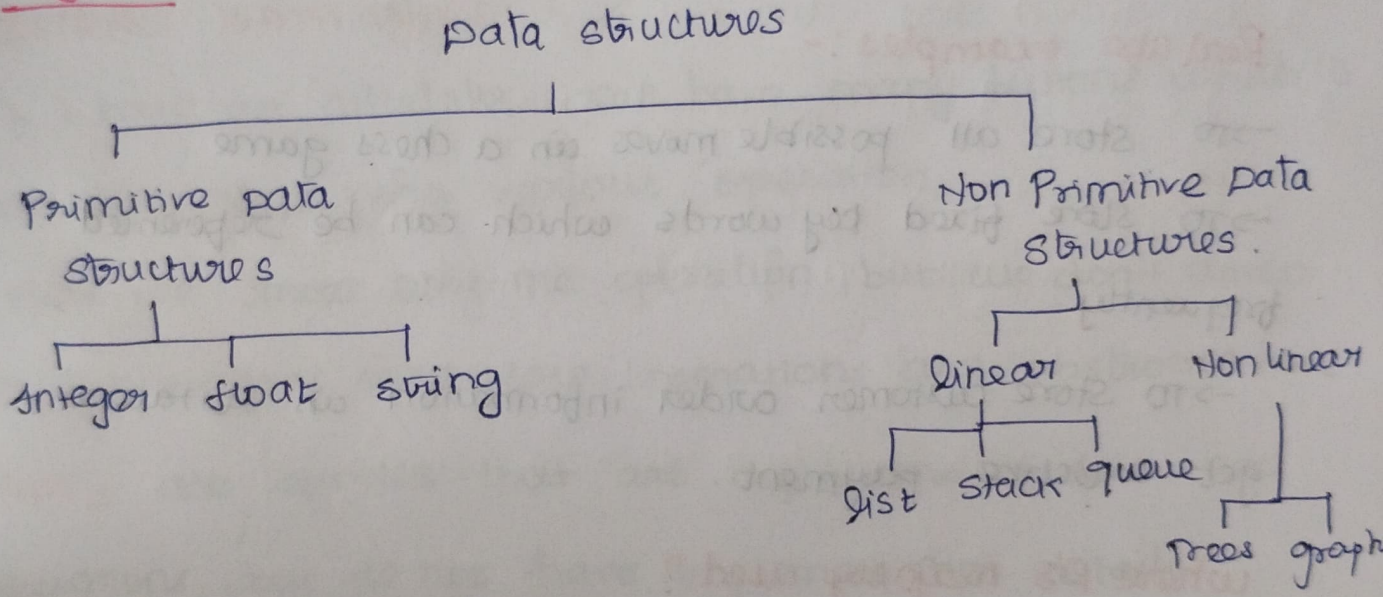
UNIT-1

LINEAR STRUCTURES AND TREES

Introduction :-

Data structure is a way of storing, organizing and retrieving data in a computer. so, that, data can be used in an effient manner. Also, it provides a way to manage large amount of data.

Types :-



Non Primitive Data structures :

These are derived from primitive data structures. This defines the group of homogeneous and non-homogeneous data items.

Ex: Arrays, lists, graphs, trees

Primitive Data Structures :-

→ It is a basic data structure

→ It directly operated upon machine instructions.

Ex : Int, char, float.

what is the importance of data structures?

-> DS provides a right way to organize the information in the digital space to analyze all larger data.

-> This is the key component of CS in AI, OS, graphics etc

-> It is the main building block for software development process. which incorporates all s/w languages

-> In IT, it is used for data processing, automated reasoning and calculation & increases problem solving ability

Real life examples :-

-> To store all possible moves in a chess game

-> To store fixed key words which can be referenced frequently.

-> To store customer order information in Restaurant to get collective payment.

where DS majorly used?

-> DS is the unique way of storing or organising data in Computer memory.

-> AI, OS, DS, Big data, Compiler design etc.,

Applications of DS :.

Search - To search an item in DS

Sort - Algorithm to sort an item in certain order.

LINKED LISTS

Linear Data structure:-

→ Here data are arranged in sequential order.

Ex: Arrays, linked lists, stacks, queues.

Non-Linear Data structure:

→ Here data are arranged in hierarchical order.

Ex: Trees and graphs.

ABSTRACT DATA TYPES:

If i have an calculator, i can have many buttons which is used for performing various operation.

Here we know only the operation, but we don't know how operations take place. operations are abstracted here.

ADT's are entities that are definitions of data, and operations, but do not have implementation details.

→ It is a set of operations., mathematical abstractions.

Ex: LIST ADT, set ADT, graph, tree ADT's.

Adv:

→ modular programming - all functions are independent to each other.

LIST ADTS

→ Array - collection of elements stored in consecutive memory location. when we are not sure of number of elements in advance, the memory usage is not efficient one.

->when elements are stored in random, then only memory usage is efficient.

->Hence there is need for data structures to remove above mentioned issues.

Linked list is the best solution.

->This does not store the elements in consecutive locations as user can add any number of items to it.

->But in an array, there is random access concepts. but linked list does not have random aces. list can be accessed in sequential manner.

Linked list:-

-> Linear collection of data elements.

-> data elements are called as node.

->This is the building block of stacks, queues.

N-size of list

A_1 - 1st element

A_N - last element

$A_1, A_2, A_3 \ldots A_{i-1}, A_i, A_{i+1}$.

Insert:- (x, i) means insert x after i.

Delete:(x) - delete x from list.

find (x) - Returns position of x

next(i) - i next it1 returns

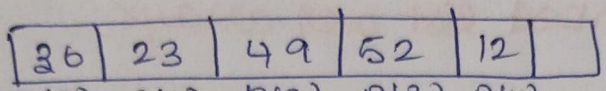
previous(i) - previous i returns

print list() - make empty()

Three types of list implementation:

1. Assay based implementation
2. Linked list
3. cursor based.

Array:



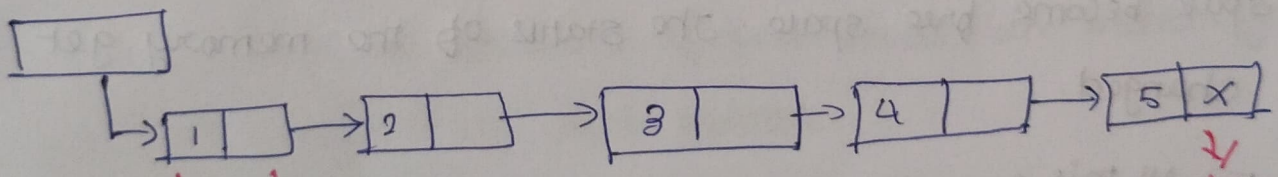
-> ordered collection of elements.

Operations:

-> Insertion.
-> deletion
-> print list : Traversal- visiting all elements in an array.
-> Find

Linked list: -> also called self referencial data type.

-> Linear collection of data elements
-> Data elements are called as node.
-> This is the building block to stacks, queues etc...

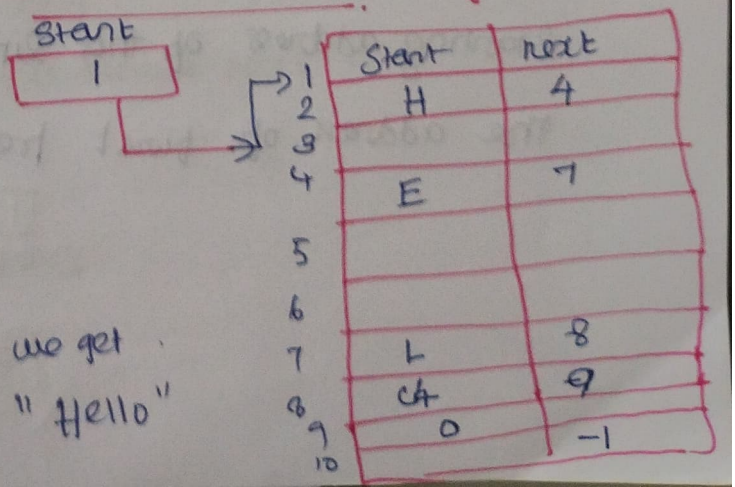


integer part
address part → pointer to next node.
Null value. (-1 in Pgm)

Code:

```
struct node
{
  int data;
  struct node * next;
};
```

Linked list in memory:



we get "Hello"

This fig has chunk of memory location where empty hold data of other application.

Here data are consecutive memory location. 1, 4, 7, 8, 10 in this case.

Advantages of linked list over arrays:-

→ linked list can be accessed in consecutive memory location and it can accessed in sequential manner.

→ linked list cannot have random access.

→ only insertion and deletion can be done at end.

→ There is no restriction of number of nodes, as like an array.

Memory allocation and de allocation for linked lists:-

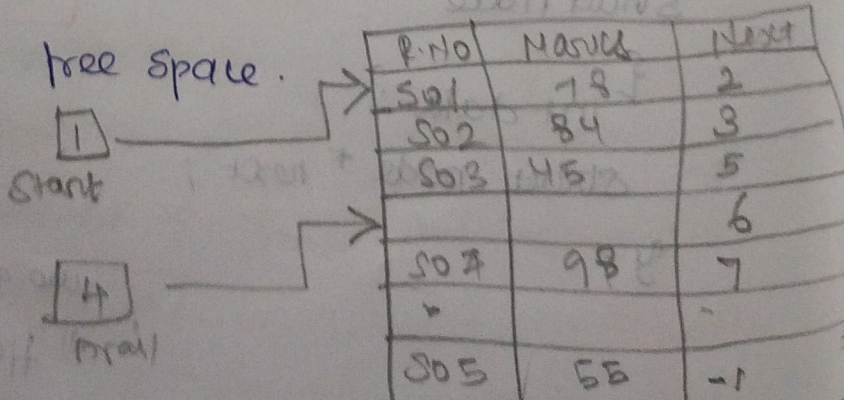
If we want to add node to an already existing linked list in the memory, we have to find a free space in the memory and then use it to store the information.

→ Thus if we delete a data means, Then available or occupied space become free space. The status of the memory get changed.

→ All this memory management is done by the OS.

→ System maintains a list of free cells called free pool.

→ A separate list is maintained for this free pool. The starting address of this linked list is "Avail" which denotes the address of first free space.



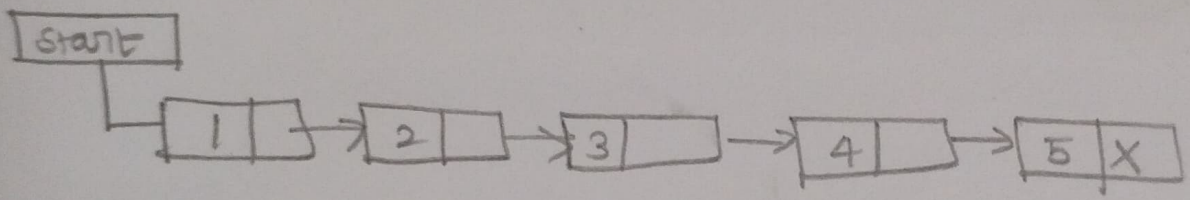
->when a node is getting deleted node means, then that space must be provided to free space.

-> This operation is done by cpu, whenever it finds free empty time or cpu is devil. or the time of task requires memory space.

->The OS must scan through memory cells and mark those cell that are being used by some other programs. Then add that free cells to free space pool.

SINGLY LINKED LISTS:-

->This is simple linked list which has data and pointer to the next node.



Traversing:-

Traversing means accessing the nodes to do some processing.

START - stores the address of first node in memory.

END Node - marked as Null pointer.

Algorithm:-

S1: Initialize SET PTR = START -> denotes 1st node a list.

S2: Repeat step S3 and 4 while PTR != NULL -> while loop until last node.

S3: PTR -> DATA -> to print 1st data

S4: Set PTR = PTR -> NEXT -> this denote next data.

End of loop.

S5: EXIT.

Singly linked list:-

It is a sequence of elements in which every element has link to its next element in the sequence.

Each element is called "Node".

Node has two fields - data field and next field.

Operations:-

Three operations can be performed on singly linked list

→Insertion

→ Deletion

→ Display.

Insertion can be done in three methods.

→Insertion at beginning of the list

→ Insertion at end of the list

→ Insertion at specific location in the list.

Insertion at beginning of the list:-

Algo:-

The following steps are used for inserting a new node at the starting of linked list.

Step 1:- Create a newnode

2:- check whether list is empty. (head == Null)

3: If it is empty, set newnode->next = Null.

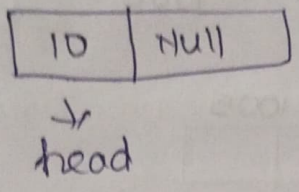
head = newnode.

4: If not empty, Newnode->next = head,

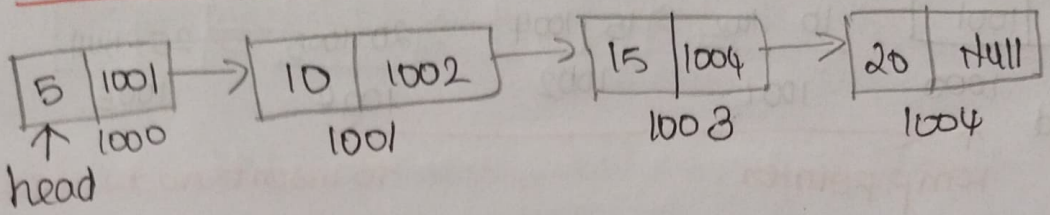
head = newnode.

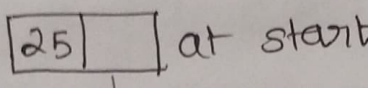
Example:-

list empty:-

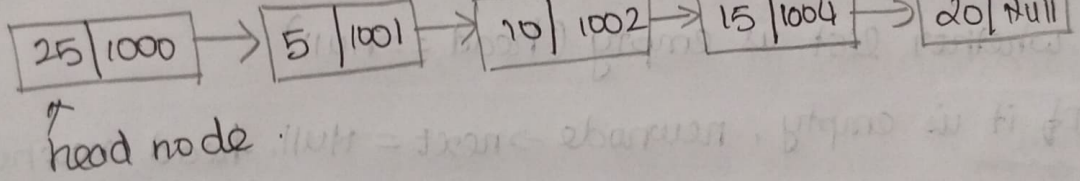


List not empty:-



To insert:-  at starting. newnode next = head

nextnode as head->next (1000), head = new (25)



Insertion at end of the list:-

Algo:-

The following steps are followed to insert the newnode at the end of the list:

list empty - Insert as head node, newnode->next = Null.

step 2: check whether it is empty (head == Null)

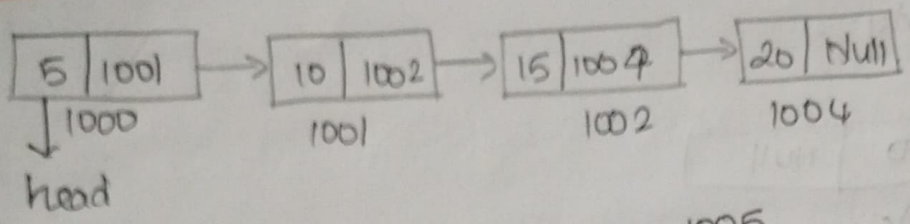
step 3: If it is empty, head = newnode, newnode->next = Null.

step 4:- If it is not empty, initialize node pointer temp and initialize with head.

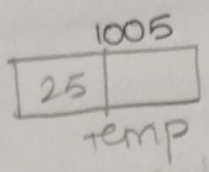
step 5: Keeping moving temp to next node until nextnode pointer becomes Null. (temp->nextnode = Null)

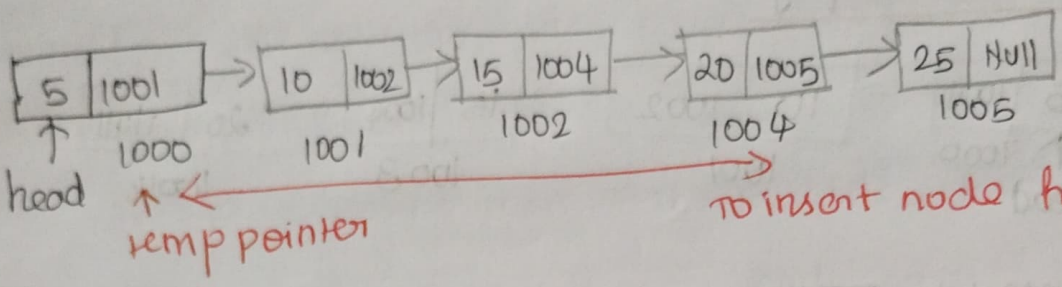
Then set temp->next = newnode, newnode->next = Null

Ex:-



Now insert | 25 | | at last.





Insertion at inbetween:-

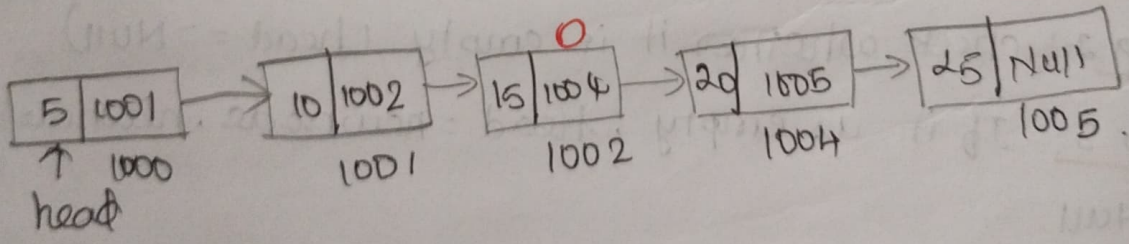
Step 1:- Create a New node.

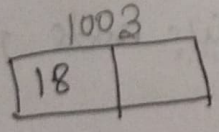
2: whether list is empty (head == null)

3: If it is empty, newnode -> next = Null, head = newnode.

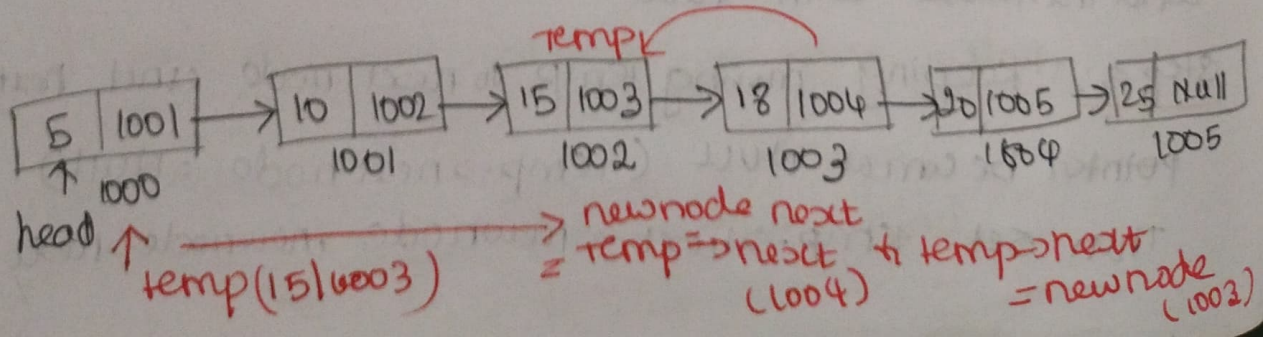
4: If not empty, define a pointer temp node, initialize with head.

5: move that temp pointer until which we want to insert the newnode (temp1 -> data) = location, which we want to insert the newnode.



Insert 

after 15 in above list.



-> But everytime moving the temp pointer, have to check whether temp is last node. If it reaches the last node, then the given node is found in the list. Hence insertion is not possible.

DELETION IN SINGLY LINKED LIST:-

In the singly linked list deletion can be done in 3 ways.

-> deletion from beginning of list

-> deletion from end of the list

-> deleting a specific Node.

Deletion from beginning of the list:-

Step 1:- Empty (head = null)

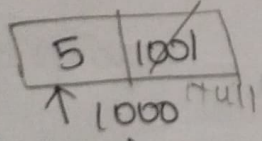
Step 2:- If it is empty, list is empty, deletion not possible.

Step 3:- If it is not empty, define pointer "temp".

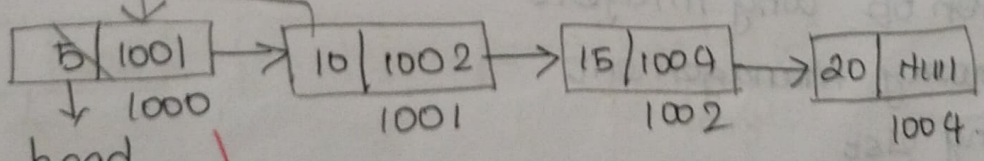
Step 4:- check whether list is having only one node which means (temp->next = null)

Step 5:- If it is True, then set head = Null and delete "temp".

Step 6:- If it is false then set head = temp->next & delete temp.

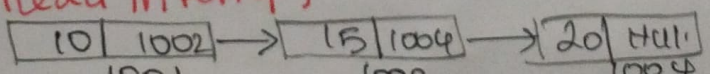


-> only one node, so temp->next -> Null
delete - 5 deleted (head = null)



head
Temp
(head in temp)

-> 1001 as head to delete head.



Deleting from end of the list :-

Step 1 :- empty (head = null)

Step 2 :- empty mean, list is empty, deletion not possible.

Step 3 :- If it is not empty, define two pointer temp 1, temp 2

temp1 = head.

Step 4 :- If list has only one node, temp1 -> next = null then

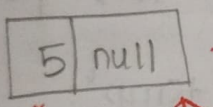
head = null & delete that node.

Step 5 :- If list is having more than one node, then set

temp 2 = temp 1, move temp1 to next node until it reaches

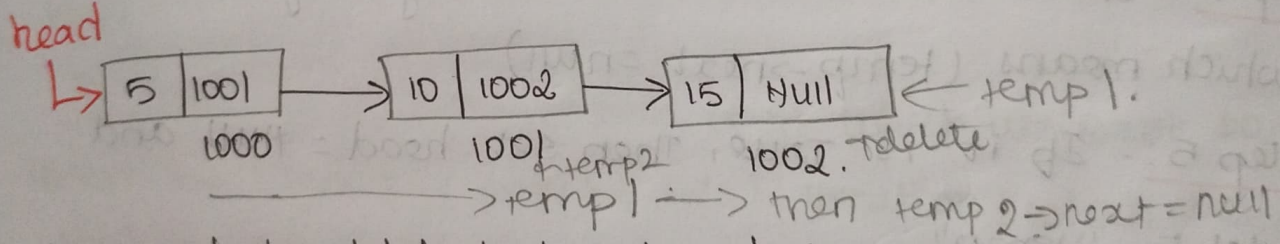
temp1 -> next = null.

Step 6 :- Then set temp 2 -> next = null, delete temp 1.



-> list has only one node.

temp1 next, delete head & make head as Null.



We want to delete last node which is 15.

temp 2 = temp 1 => temp 2 = 5 = move temp1 to 15.

Applications :-

-> used to implement stacks and Queues.
-> polynomial manipulation
-> addition of long integers
-> memory management
-> mailing list
-> Allocation of files in a memory.

DOUBLY LINKED LIST:

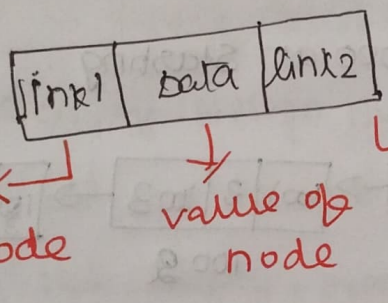
Drawback of singly linked list:-

→ we cannot traverse back to the previous node.

Definition:-

Double linked list is a sequence of elements in which every node has link to the previous node and to the next element in the sequence.

Hence the traversing forward can be done by next field and backward by prev field.



→ here firstnode always points to head

→ prev pointer of first node must be Null.

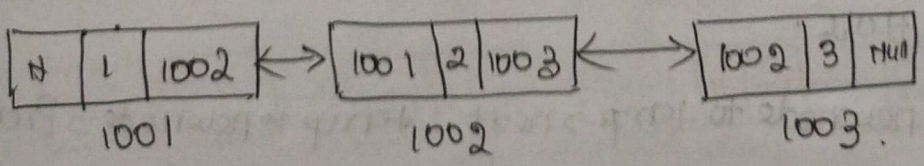
→ Next field of prev last node must be Null.

operations on doubly linked list:-

The following operations can be done

1. Insertion
2. deletion
3. display.

Ex:



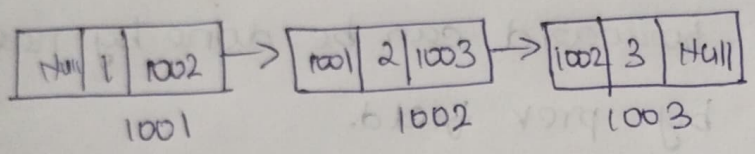
Insertion at the beginning of list:-

Step1:- Create Newnode, make Newnode->previous as NULL.

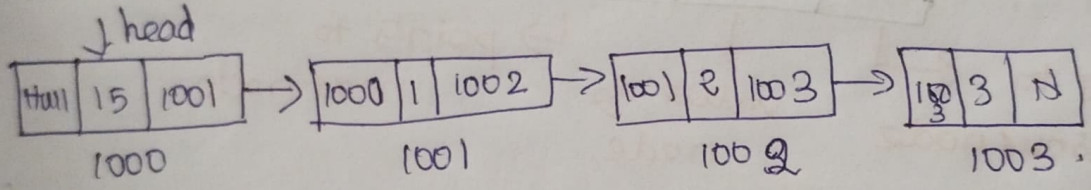
Step 2:- Check whether list is Empty (head==NULL)

Step 3:- If it is empty, then assign NULL to newnode->next and newnode to head.

Step 4:- If it is not empty, assign head to newnode->next and Newnode to head.



Insert 15 with address 1000 at Starting.



Insertion at last in list:-

Step 1:- Create Newnode, newnode->next as Null.

step2:- Empty is list hence (head==Null)

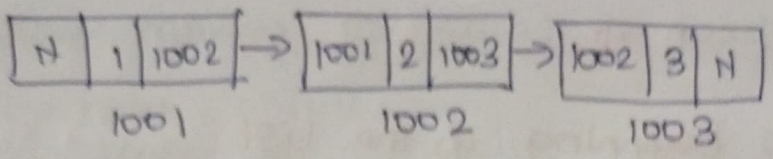
Step 3:- If it is empty, then assign Null to newnode, make Newnode->next as Null, Newnode to head.

Step 4:- If it is not empty, define node pointer temp and initialize as head.

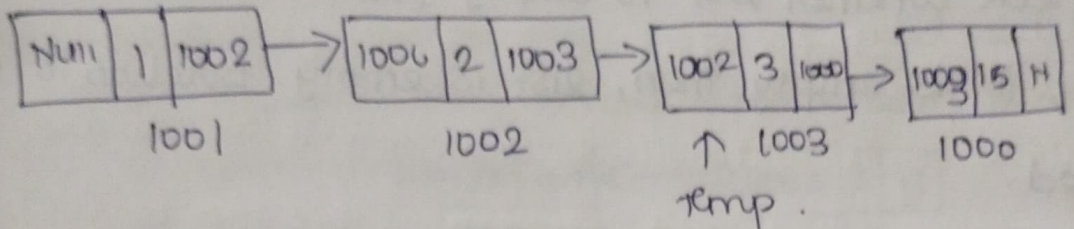
Step 5:- Keep move temp pointer until it reaches the last node temp->next = NULL.

Step6:- Assign newnode to temp->next, temp to newnode->Prev

Ex:-



Insert 15 with 1000 at end.



Insertion at inbetween after needed Node :

Step 1:- Create newnode with given value.

Step 2:- If list is empty (head == Null)

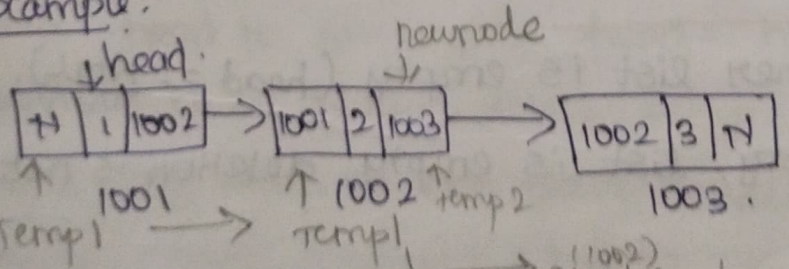
Step 3:- If it is empty, Assign Null to both newnode->prev & newnode->next and set Newnode to head.

Step 4:- If it is not empty, create 2 pointers temp1 and temp2, initialize temp1 with head.

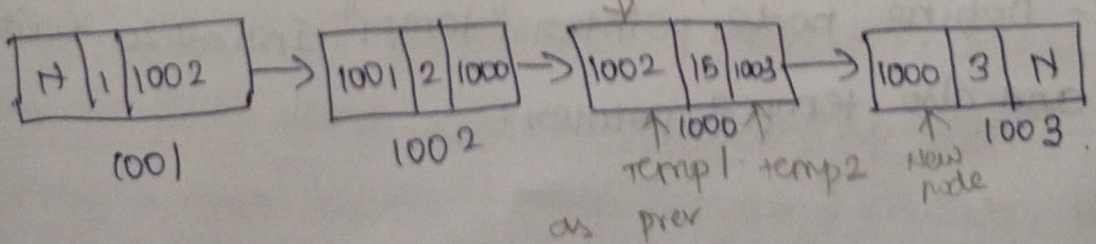
Step 5:- Keep move temp1, until it reaches the location where it want to insert newnode.

Step 6:- Assign temp1->next to temp2, newnode to temp1->next to Newnode->prev, temp2 to Newnode->next, Newnode to temp2->prev.

Example:-



Now insert 15->1000 after 2



Deletion:

Deleting from beginning of the list:

Step 1:- check whether list is empty (head == NULL)

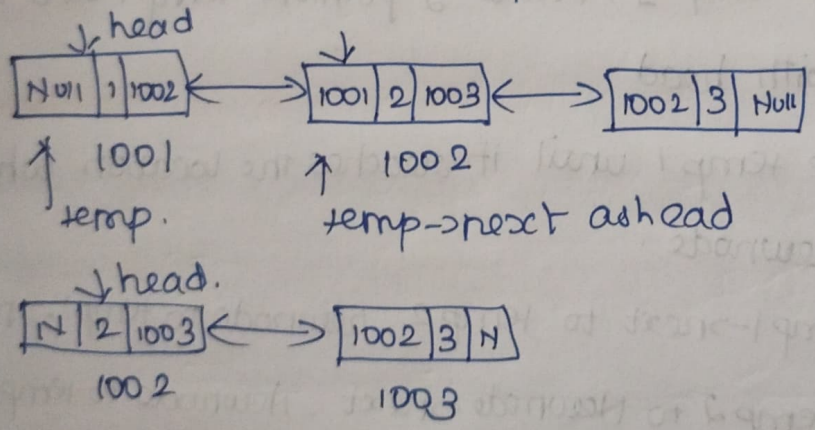
Step 2:- If it is empty then, list is empty. deletion is not perpormed.

Step 3:- If it is not empty, then define Node pointer 'temp' and initialize with head.

Step 4:- check whether list is having only one node. (temp->previous = temp->next)

Step 5:- If it is TRUE, set head = NULL, delete temp.

Step 6:- If it is FALSE, temp->next to head, NULL to head->previous and delete temp.



Deleting from end of the list:

Step 1:- check whether list is empty (head == NULL).

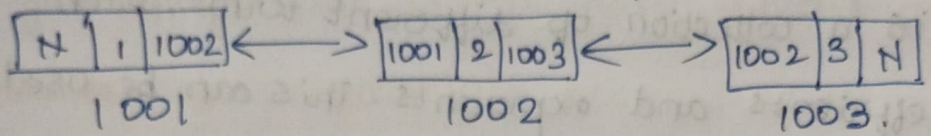
Step 2:- If it is empty, list is empty, deletion is not possible.

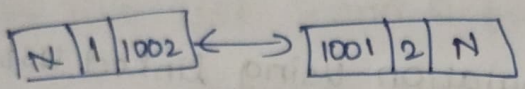
Step 3:- Define node pointer "temp1". Initialize with head and also temp 2 pointer.

Step 4: Keep moving the temp1 to reach last node where last node -> next == NULL.

Step 5: If it is true, delete temp.

Step 6: Assign Null to temp -> next and delete temp.
make Null to prev -> temp -> next and delete temp.





-> deleted last node.

Deleting a specific node from the list:

Step 1: check whether list is empty (head == NULL)

Step 2: If its empty, list is empty, deletion is not possible.

Step 3: If it is not empty, create "temp" - pointer.

Step 4: Keep move the temp, until it reaches "temp" -> Not given node to be deleted.

Step 5: If it reaches last node, given node is not found.

Step 6: If it has one node, set head = NULL, delete temp.

Step 7: If it has multiple node, check whether temp is the first node, (temp == head). If temp is head, then move head to next node, head = head -> next, head of previous to be Null., delete temp.

Step 8: If temp is not first, check whether it is the last node (temp -> next == NULL)

Step 9: If temp is last node, tem -> prev -> next = NULL.

Step 10: else, temp -> prev -> next = temp -> next, (temp -> next -> prev = temp -> prev), delete temp.

POLYNOMIAL MANIPULATION

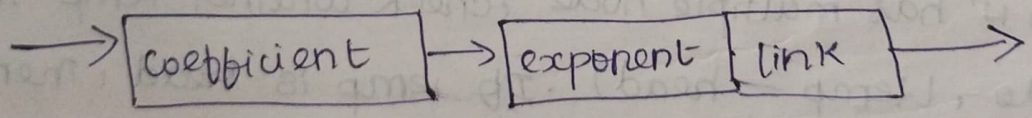
It is the most important application of linked list. This polynomial is the important part of mathematics which is supported by data type by most languages. A polynomial is a collection of different terms, each comprising coefficients and exponents. This can be used in representing linked list.

Representing polynomial manipulation using linked list:

-> Node are used for representing polynomial manipulation.
-> No two terms should have same exponents.
-> No term has zero coefficient
-> If no coefficient means, take a value of 1.

Node has 3 parts:-

-> First part contains the value of coefficient of term
-> second part contains the value of exponent
-> Third part, link points to the next term (next node).

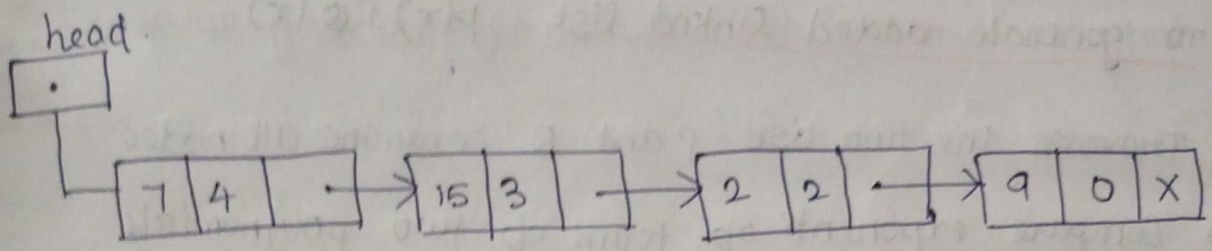


Example:- Consider a polynomial

$p(x) = 7x^4 + 15x^3 - 2x^2 + 9$.

here 7, 15, -2, 9 are coefficients 4, 3, 2, 0 are the exponents.

The linked list represents of above polynomial is



hence here number of nodes is equal to number of terms in the given polynomial. so we have 4 nodes.

Addition of polynomials :-

consider two list p and Q four addition. First traverse through p and Q. we have to take corresponding terms and have to compare their coefficients.

->If two exponents are equal, add the coefficients to create a new coefficient. If new coefficient is zero, then drop.

->If it is not zero, the node is inserted at the end of new linked list of resulting polynomial.

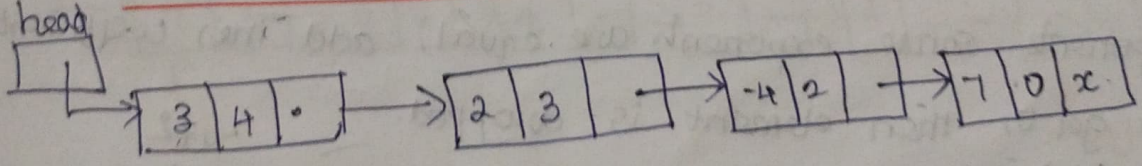
->If one of the exponent is larger than other, then the corresponding term is immediately placed to newlinked list

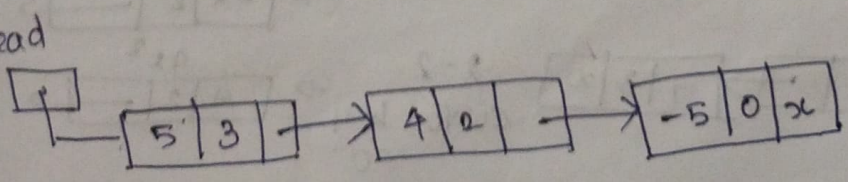
-> If it is smaller, compare with next term from other list.

->If one list ends before other, the rest of terms of the longer list is inserted at the end of new linked list.

Ex:- $p(x) = 3x^4 + 2x^3 - 4x^2 + 7$

$Q(x) = 5x^3 + 4x^2 - 5.$ (+) add.



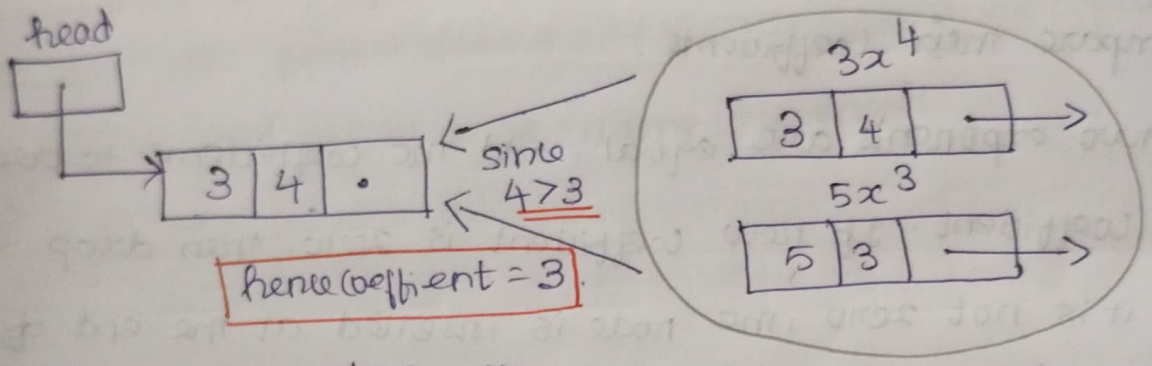


To generate added linked list $p(x) + Q(x)$.

① → Traverse the two lists p and Q, examine all nodes.

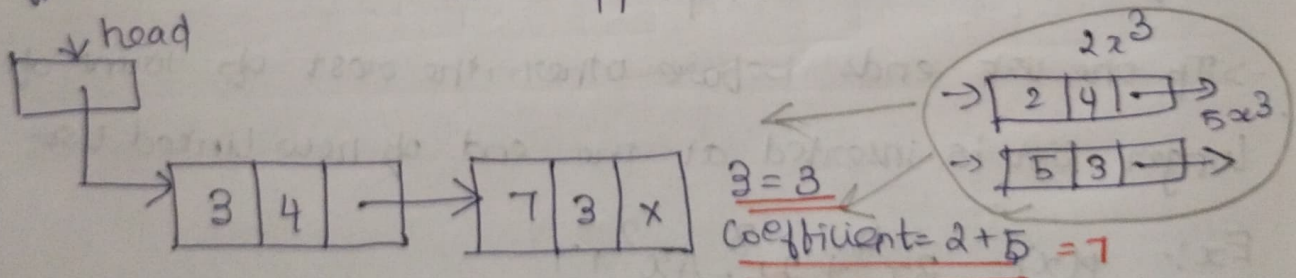
→ Compare exponents of terms of two polynomials.

② Here the first term of polynomials p and Q contains the exponents 4 and 3 respectively. Since the exponent of 1st term of polynomial p is greater than other polynomial Q. So, the term having larger exponent is inserted to the new list. where the new list looks like below.

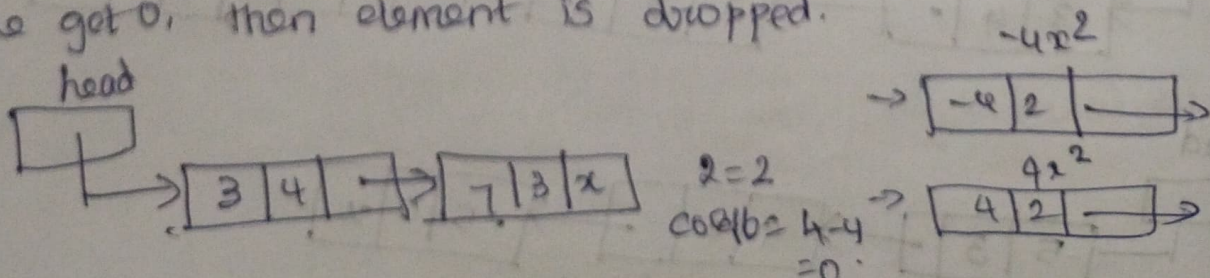


Hence coefficient 2 with exponent 4 is added.

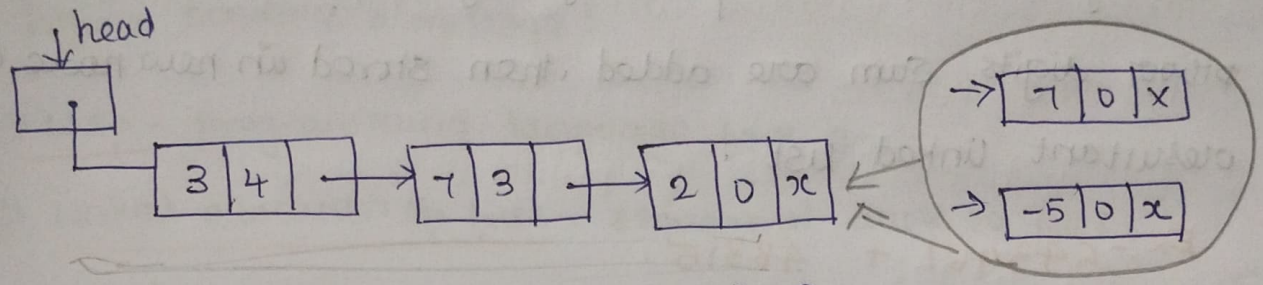
③ → we then compare the exponent of next term of p with current Q. Here now, two exponents are equal, so their coefficients are added and appended to new list.



④ → move to the next term of p and Q, compare their exponents. Since exponents are equal, add their coefficients, we get 0, then element is dropped.



(5)
→ move to next term of two lists. P and Q, we find that the corresponding terms have same exponents equal to 0. we add their coefficient and append them to new list.



0 = 0

Coefficient = 7 - 5 = 2

↓

LL of resulting polynomial.

Addition of long positive integer using linked list:

→ Max value of largest integer 32767.

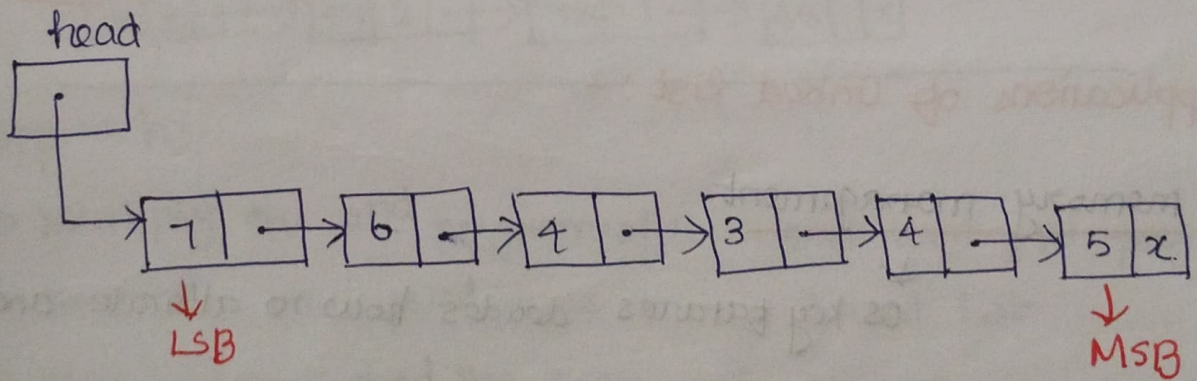
→ cryptography need storing and manipulating larger integer in security algorithms.

→ In such case, linked list can be used.

→ digits of longer integer must be stored from right to left in a linked list, so first node contains least significant digit. last node contains most significant bit.

Ex: 543467 can be represented as below

MSB LSB



For performing the addition of two long integers.

→ Traverse two linked lists in parallel from left to right

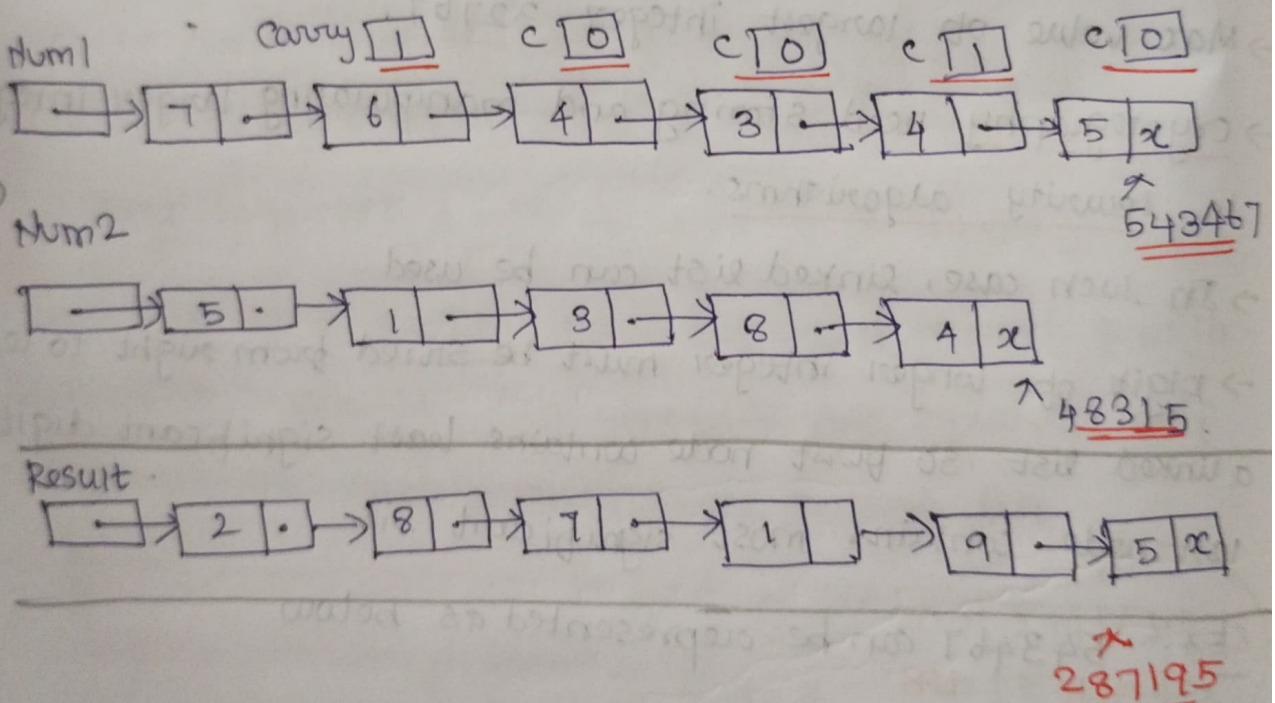
→ during traversal, corresponding digits and a carry from prior digits sum are added. then stored in new node of resultant linked list.

Ex:- 543467 + 48315.

543467 - first node is pointed by Num1 pointer.

Similarly represent 48315 as second linked list. Hence first node of second linked list is Num2 pointer.

Add those and store in new linked list.



Applications of linked list :-

→ memory management

↓

OS key features. decides how to allocate and reclaim storage of processes running on the system. to keep track of position of memory allocation available position.

→ Mailing lists :-

Linked lists use their use in email applications. As linked list is difficult to predict multiple lists may be later before sending a message.

→ Lisp - programming language for AI.

→ Linked allocation of files - sequential access to files.

→ Virtual memory.

→ Stacks, queues, hash tables and graphs.

POLYNOMIAL MULTIPLICATION :

Poly 1 :: $3x^2 + 5x^1 + 6$

Poly 2 :: $6x^1 + 8$.

$8(3x^2 + (5x^1 \times 6x^1))$

$\Rightarrow 3x^2 \cdot (8) + (30x^2)$

$= 24x^2 + 30x^2$

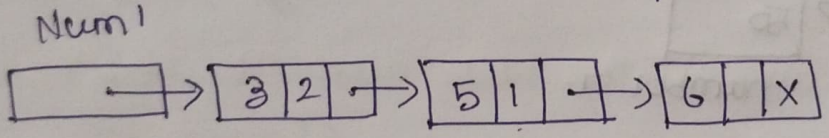
$= 54x^2$

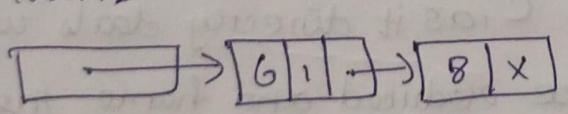
$5x^1(8) + 6(6x^1)$

$= 40x^1 + 36x^1$

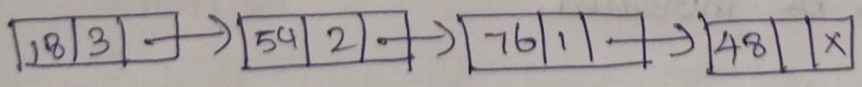
$= 76x^1$

$18x^3 + 54x^2 + 76x^1 + 48$





×



Approach :-

→ Multiply the 2nd polynomial with each term of 1st polynomial

→ Store the multiplied value in new linked list

→ Then we will add the coefficients of element having same powers in the resultant polynomial.

Pointers in C:- [For DAA DS programs]

Pointer is a variable which stores the address of another variable. pointer variable can be int, char, array, function or any other pointer.

32 bit architecture - size of pointer is 2 bytes.

Ex:- int n=10;

int *p=&n;

here p is the pointer variable which stores of address of variable 'n'.

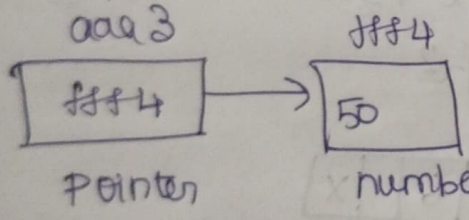
↓

indirection pointer, dereference pointer.

int *a; // pointer to int

char *a; // pointer to char.

↑address of pointer variable p.



Pointers are important in C, as it directly deals with memory, hence code can be reduced and hence the performance can be increased.

→ Greatest advantage of C, as java, python does not support pointers.

structures in C:

→ way of grouping several elements with diff data types into one place.

→ Each variable in a structure is called structure member.

→ created by using struct → keyword

Ex:

```
struct mystructure // declaration
{
  int mynum;  //int variable
 Char myname; // char variable.
};  end with semi colon.
```

TO access the structure member. we have to create a variable of it.

```
struct mystructure
{ int mynum;
  Char, myname;
};
int main()
{
struct mystructure s1; -> we can also create another variable (struct mystructure s2;)
return 0;
}
```

(x) Note.

Structure members can be accessed by dot operator (.)

```
int main()
{
struct mystructure s1; // structure variable creation.
s1.mynum = 10; // Assigning values
s2.myname = 'Ax';
printf("my num is : %d", s1.mynum); // printing values.
}
```

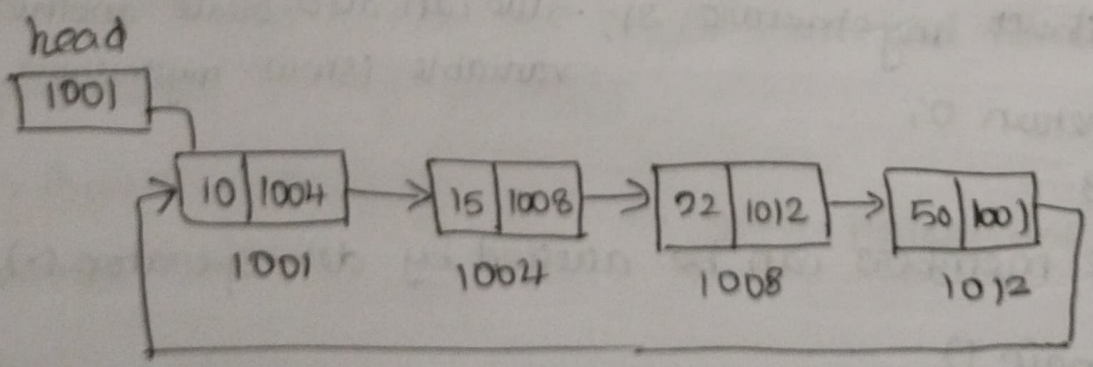
for string variable, we have to use strcpy function

```
struct mystructure
 {
  int mynum;
  char mystring[30];
 };
int main()
 {
  struct mystructure s1;
 strcpy(s1.mystring = "hello");
 printf("string is": %d", s1.mystring);
 }
```

CIRCULAR SINGLY LINKED LISTS:

A circular linked list is a sequence of elements in which every element is has a link to its next element in the sequence where the last element has the link to the first element.

Ex:-



Operations:-

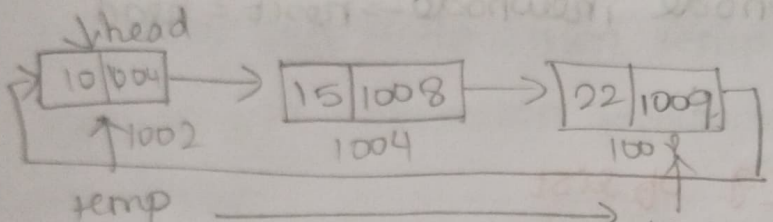
-> Insertion

-> Deletion

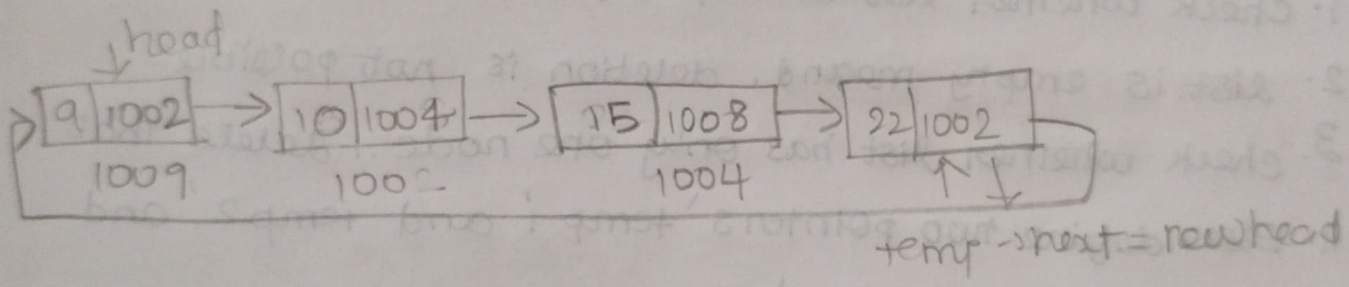
-> Display.

Inserting at the beginning of the list:-

1. Create newnode with given value.
2. check whether list is empty (head == NULL)
3. If list is empty, set head = newnode, newnode->next = head.
4. Else, create temp pointer, initialize with head.
5. Keep moving temp until temp->next = head.
6. Step newnode->next = head. head = newnode, temp->next = head



Now insert 9- ~~1004~~ 1009 as 1st node.



Inserting at specific location:-

1. Create newnode with given value.
2. If it is empty, set newnode = head, newnode->next = head.
3. Else, create a temp pointer, move temp until it reaches the node after we want to insert the newnode.
4. If temp reaches the last node, then given node is not found.
5. If temp reaches exact location, check whether temp->next == head.
6. If temp->next = head, means this last node. set temp->next = new, newnode -> next = head.

7. Else newnode->next = temp->next , temp->next = newnode.

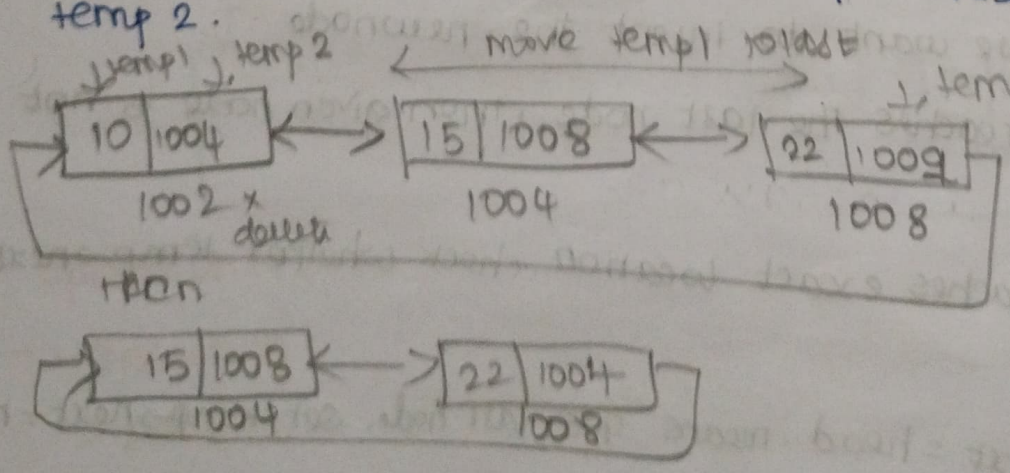
Inserting at end of the list:-

1. Create new node with given value.
2. If list not empty ,set head = newnode. newnode->next = head.
3. If ist not empty. initialize temp pointer , move temp until it reaches , temp->next == head.
4. set temp->next = newnode , newnode->next = head.

Deletion:

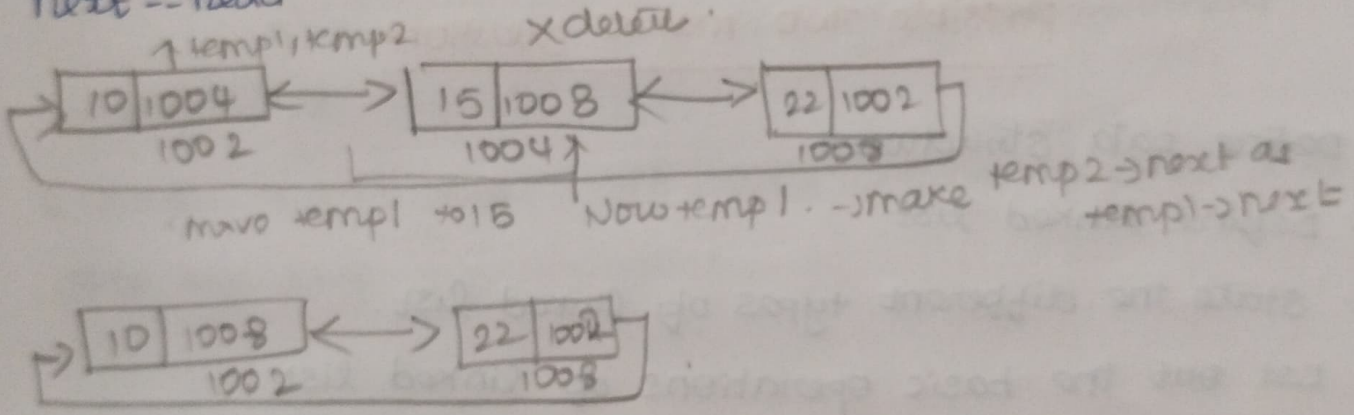
Deletion from beginning of list

1. Check whether list is empty (head == NULL)
2. list is empty means, deletion is not possible.
3. Check whether list has only one node., delete that
4. Else, define two pointers `temp1' and `temp2' and initialize both with head.
5. If it has only one node, temp1->next == head.
6. set head == NULL , delete 'temp1'.
7. else, move temp1 to last node., temp1->next == head.
8. set head = temp2->next, temp1->next = head, delete temp2.



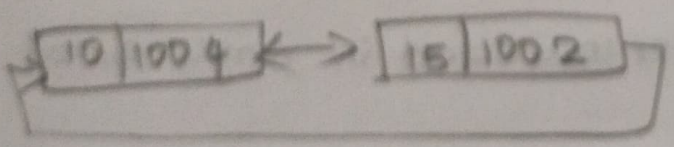
Deletion at Specific node:

1. Check all predefined condition.
2. If list is not empty, define temp1 and temp2 pointer, with head.
3. Keep move temp1 to node which we want to delete and make temp2 = temp1.
4. If list has only one node, make (temp1->next == NULL)
5. If else, if deleting node is first node, set temp2 = head. head = head->next, temp2->next = head. delete temp1.
6. If it is last node, then temp1->next == head.
7. Else, if it is inbetween node, temp2->next = temp1->next == head



Deleting last node:-

1. If list has only one node, then head = NULL delete head.
2. Else temp2 = temp1, move temp1 to next node until temp1->next = head
3. Now set temp2->next = head, delete temp1.



Applications of doubly linked list:-

-> It is used by browser to implement backward and forward, Navigation of pages

->To represent a classic game deck of cards.

->To do undo and redo operations

-> For constructing MRU / LRUU cache.

Applications of Circular linked list:

->To implement stack and queues

-> used in Computer Networking for token scheduling

-> For CPU Scheduling.

Possible TWO marks questions

1. Define Data structures.
2. Define linked list
3. State the different types of linked list.
4. List out the basic operations of linked list.
5. Difference between arrays and linked lists.
6. Define Abstract Data type (ADT).
7. what do you mean by non linear data structures? Ex.
8. what is linear data structures? Example.
9. List out the applications of Data structures.
10. what are the pitfalls encountered in singly linked list.
11. why linked list is used for polynomial arithmetic?
12. what is polynomial manipulation.

16 marks :

1. Explain insertion operation in singly linked list. How nodes are inserted after a specific node.
2. Explain all insertion operations in doubly linked list. with example.
3. Explain all deletion operations in doubly linked list with example.
4. Explain insertion, deletion operation in circular linked list with neat sketch.
5. Add $p(x) = 3x^4 + 2x^3 - 4x^2 + 7$ and $Q(x) = 5x^3 + 4x^2 - 5$ using polynomial manipulation linked lists.
6. Add 327676 and 543467 using linked list in step by step explanation.