
Data
Structures
Fibonacci Heap
FM-Index
Hash Table
Trie
Top Tree
Splay Tree
Beap
Tree
Heap
BST
queue
Soft Heap
Stack
Array
Binary Heap
Soft Heap
Weak Heap
Suffix Tree
Doubly Linked List
Stack
Trie
Beap
Rope
BIT
BST
Tango Tree
array
Queue
Array
Trie
Treap
Binomial Heap
Tree
Array
Trie
ডেটা স্ট্রাকচার শিখি বাংলাতে...
Queue
Beap
Rope
Heap
Treap
BIT
queue
Suffix Array
BST
Tree
Splay Tree
Soft Heap
Linked List

Singly linked list

SINGLY LINKED LIST: implementation of a node

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

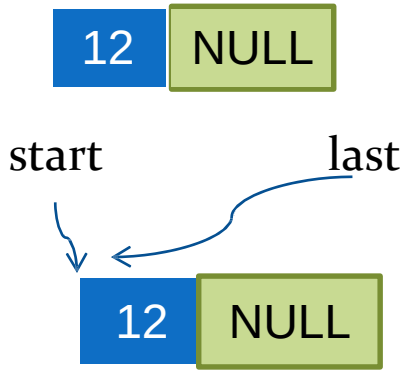
The address of the next node

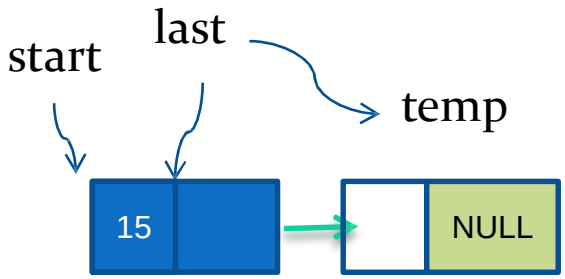
Creation of Linked List

```
class linked_list
{
        private:
                node *start; node * last;
        public:
                linked_list()
                 { start=NULL; last= NULL;}

                void add_node(int value);
                void insert_first(int value) ;
                void insert_at_position(int pos,int val);
                void delete_first();
                void delete_last();
                 void delete_at_position(int pos);
                void display();
};
```

Linked List:add nodes forward

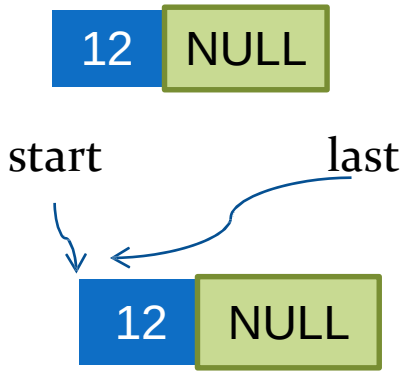
```
void linked_list:: add_node(int value)
    {
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        if (start == NULL)
        {
                start=temp;
               last=temp;
        }
        else
        {
          last->next = temp;

         last = temp;

        }
    }
```

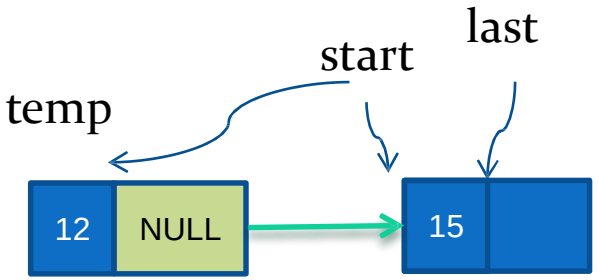




Linked List:insert_first

```
void linked_list::insert_first(int value)
{
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        if (start == NULL)
        {
                start=temp;
               last=temp;
        }
    else{
        temp->next=start;
        start=temp; }
}
```





insert a node in between a linked list

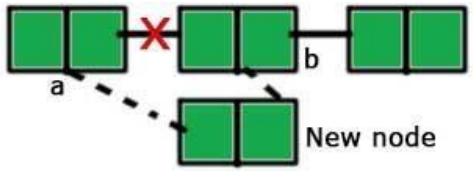
- The steps for inserting a node between two nodes a and b:

1. Make a new node

   node *tmp=new node

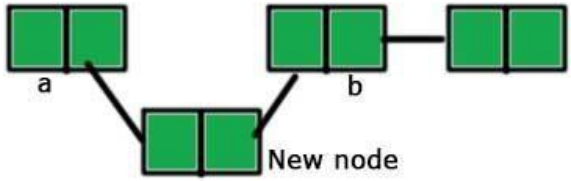
2. Point the 'next' of the new node to the node 'b' (the node after which we have to insert the new node). Till now, two nodes are pointing the same node 'b', the node 'a' and the new node.

   temp->next=b



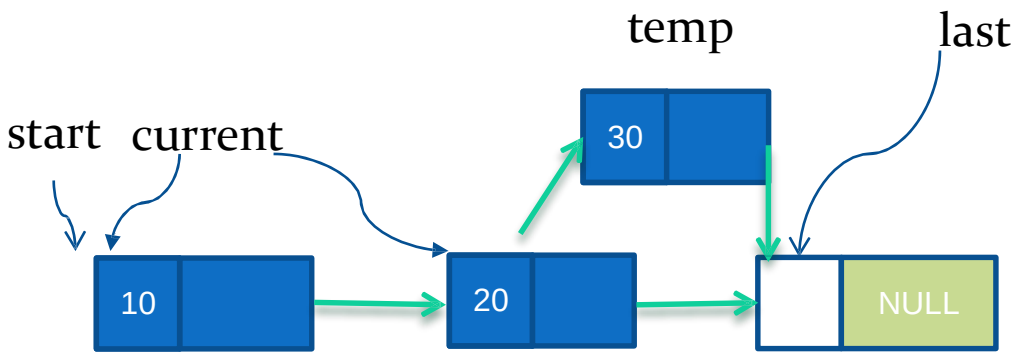
3. Point the 'next' of 'a' to the new node.

   a->next=temp



Linked List; insert_position

```
void linked_list::insert_at_position(int pos, int value)
    {node *temp=new node;
    temp->data=value;
    temp->next=NULL;
node *current=start;
    for(int i=1;i<pos-1;i++)
    {
          current=current->next;
          if(current==NULL)
     {cout<<" the list has element lessthan
                              "<<pos<< "element ";return;}
    }
    temp->next=current->next;
    current->next=temp;
}
```



Linked List:display

```
void linked_list:: display()
{
        if(start==NULL) {cout<<"list is empty ";return;}
        node *temp=start;

        cout<<" the elements of the list are    "<<endl;

        while(temp!=NULL)
        {
                cout<<temp->data<<"\t";
                temp=temp->next;
        }

}
```

Delete last element in linked list

- Check if the list is empty

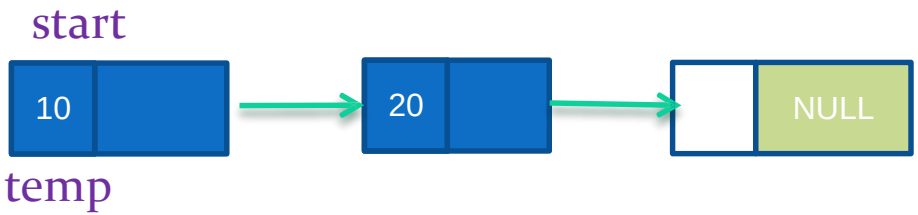
  if(start==NULL)
- If it not empty check if it has one element only

  if (start->next==NULL)
- If it has more than one element

Delete first element in linked list

- Copy the address of first node i.e. start node to some temp variable say temp

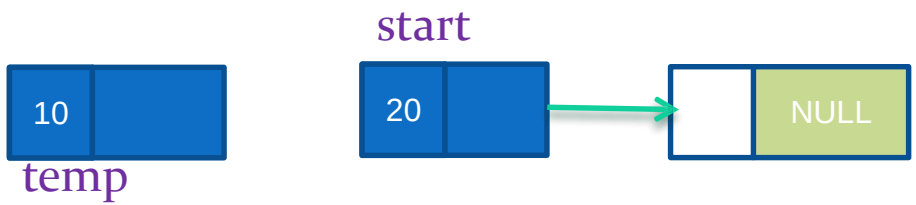
  node *temp= start;



- Move the start to the second node of the linked list i.e.

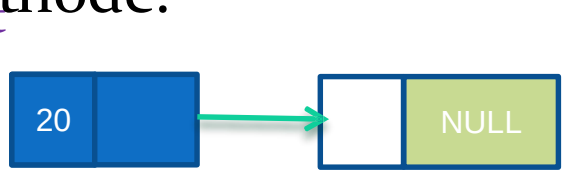
  start=start->next;

- Disconnect the connection of first node to second node.



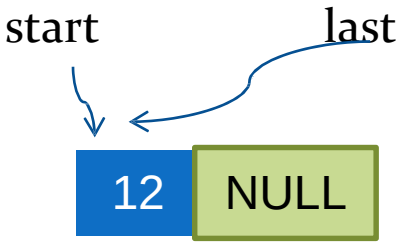
- Free the memory occupied by the first node.

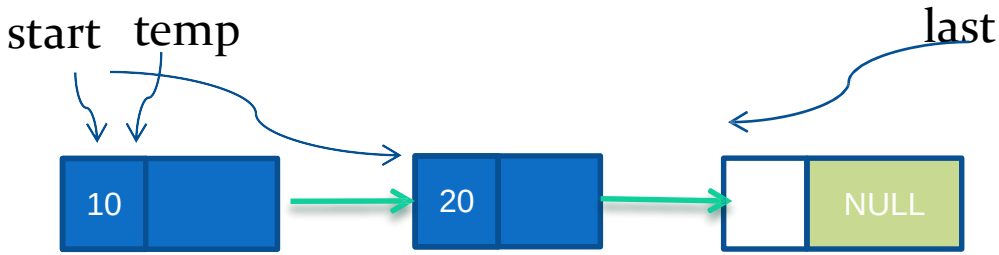
  delete temp



Linked List :delete_first

```
void linked_list::delete_first()
{//list is empty
    if(start==NULL){ cout<<"the list is empty"
    ; return;}
    //only one node
    if (start->next==NULL)
    { delete start; delete last; return;}
    //more than one node
    node *temp= start;
    start=start->next;
    delete temp;
}
```





Delete last node from linked list

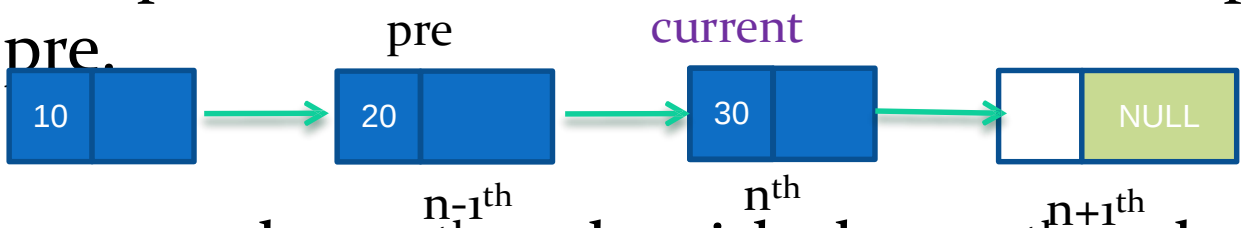
- Traverse to the last node of the linked list keeping track of the second last node in some temp variable say current .
- disconnect the second last node with the last node i.e. current->next = NULL.
- Free the memory occupied by the last node.

Linked List: delete_last

```
void linked_list::delete_last()
 {// list is empty
            if(start==NULL) { cout<<"the list is empty" ; return;}
//only one node
             if (start->next==NULL)
     { delete start; delete last;return;}

//more than one node
            node *current=start;
            node *temp=last;
            while(current->next!=last)
            { current=current->next; }
            current->next=NULL;
            last=current;
            delete temp;
 }
```

Delete element at specific position

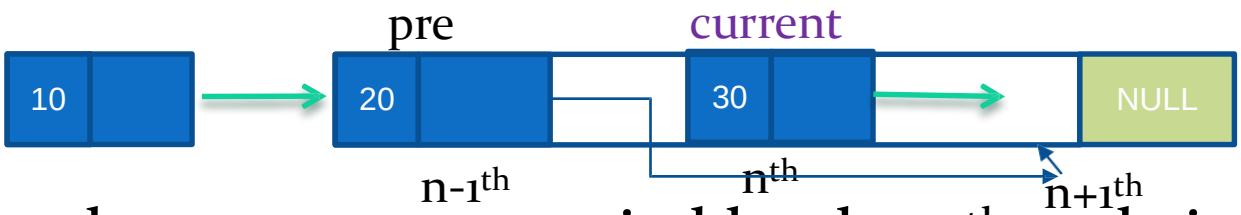
- Traverse to the n^{th} node of the singly linked list and also keep reference of $n-1^{th}$ node in some tempvariable say pre.



- Reconnect the $n-1^{th}$ node with the $n+1^{th}$ node

i.e. pre->next=current->next;

(Where pre is $n-1^{th}$ node and current node is the n^{th} node and current->next is the $n+1^{th}$ node).



- free the memory occupied by the n^{th} node i.e. current node.

Linked List: delete_ position

```
void linked_list::delete_at_position(int pos)
{
    node *current=start;
    node *pre=start;
    for(int i=1;i<pos;i++)
     { pre=current; current=current->next;
          if(current==NULL) {cout<<" the list has element lessthan"
                                        <<pos<< "element "; return;}
    }
     pre->next=current->next;
    delete current;
}
```

Main function

```
int main()
{
    int value; linked_list list;
    cin>>value;
    list.add_node(value);
    list.add_node(10);
    list.insert_first(4);
    list.insert_first(5);
    list.insert_first(6);
    list.display();
    list.insert_at_position(3,-1);
    list.display();
    list.delete_at_position(3);
    list.display();

    return 0;
}
```

Applications of linked list

linked list applications

- Dynamic Memory Management. In allocation and releasing memory at runtime
- Use linked list as a stack by adding and removing elements from the beginning of the list.
- Use linked list as a queue by adding in the beginning, removing from the end

linked list applications

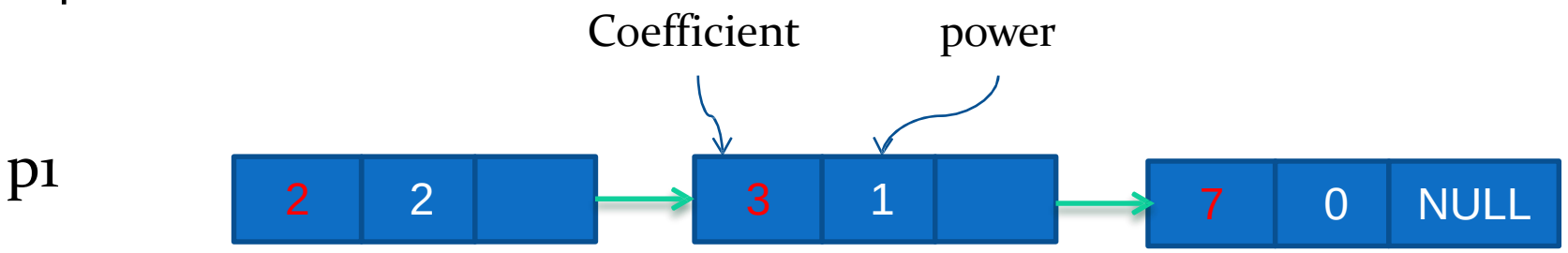
- A great way to represent a deck of cards in a game
- Graphs? Adjacency list representation of graph.
- In hash table , each bucket of the table can itself be a linked list (chaining)
- Use linked list for addition /subtraction /multiplication of two polynomials. Eg:

  $p1=2x^2+3x+7$ and $p2=3x^3+5x+2$

  $p1+p2=3x^3+2x^2+8x+9$

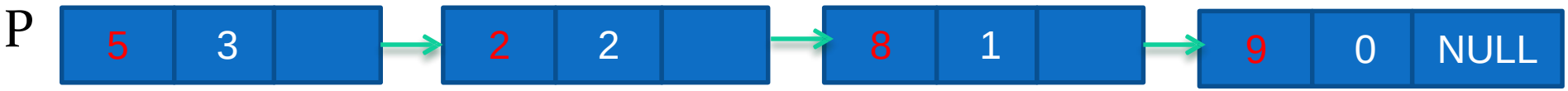
Polynomials using Linked List

$p1=2x^2+3x+7$



$p2=5x^3+5x+2$

$P= p1+p2=5x^3+2x^2+8x+9$



Polynomials using Linked List: node structure

node


```
struct node
{
        int coeff;
        int pow;
        node *next;

}
```

Assignment 1

- write a program to add two polynomials. using linked list as representation of polynomials
- Deadline 2-11-2019


THANK
YOU


Any
Question?