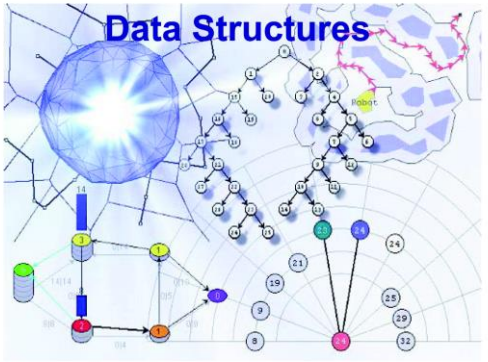
# BBM 201
# DATA STRUCTURES

**Lecture 8:**

Linked Lists Examples




**2017-2018 Fall**

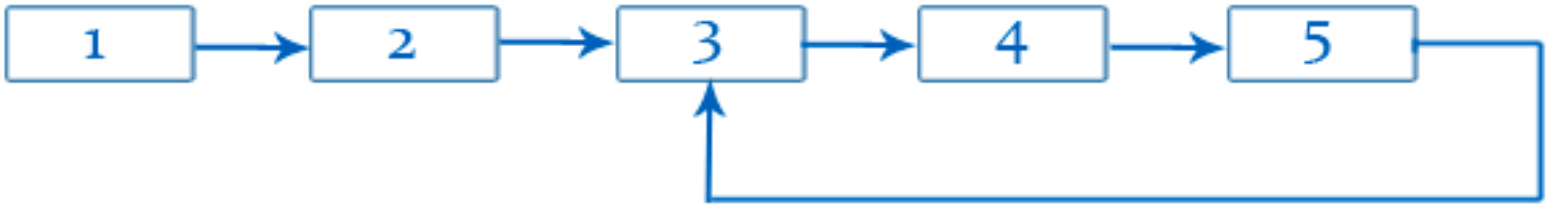
## Q1 : Delete alternate nodes of a Linked List

Given a Singly Linked List, starting from the second node delete all alternate nodes of it. For example, if the given linked list is 1->2->3->4->5 then your function should convert it to 1->3->5, and if the given linked list is 1->2->3->4 then convert it to 1->3.

# Q1 : Delete alternate nodes of a Linked List

```
struct node *deleteAlternative(struct node *head)
{
   if(head==NULL)
      return NULL;
   struct node *p = head;
   struct node *q = p->link;
   while( p != NULL && q != NULL)
   {
      p->link = q->link;
      free(q);
      p = p->link;
      if( p != NULL)
         q =  p->link;
   }
   return head;
}
```

## Q2 : Write a program to detect loop in a Linked List.



# Q2 : Write a program to detect loop in a Linked List.

```
void detectLoop(struct node *head)
{
    struct node *slowPtr,*fastPtr;
    slowPtr = head;
    fastPtr = head;

    while(slowPtr != NULL && fastPtr != NULL && fastPtr->next != NULL)
    {
       slowPtr = slowPtr->next;
       fastPtr = fastPtr->next->next;
       if (slowPtr == fastPtr)
       {
           printf("%s","Loop Detected");
           exit(0);
        }
    }
}
```

**Time Complexity** : O(n)
**Space Complexity** : O(1)

**Q3 : Write a C program to return the nth node from the end of a linked list.**

**Q3 : Write a C program to return the nth node from the end of a linked list.**

**Brute Force Approach:** Let's assume length of linked list be 'L' , nth node from end = L – n + 1 node from start.

Step 1: Iterate over the linked list and determine its length.

Step 2: Now using the above mentioned formula , we can determine the position of the required node.

Step 3: So we iterate the linked list again till we reach the required node.

Step 4: Return the required node.

**Q3 : Write a C program to return the nth node from the end of a linked list.**

**Solution 2** : Suppose one needs to get to the 6th node from the end in this LL. First, just keep on incrementing the first pointer (ptr1) till the number of increments cross n (which is 6 in this case)

Step 1 : 1(ptr1,ptr2) -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10

Step 2 : 1(ptr2) -> 2 -> 3 -> 4 -> 5 -> 6(ptr1) -> 7 -> 8 -> 9 -> 10

Now, start the second pointer (ptr2) and keep on incrementing it till the first pointer (ptr1) reaches the end of the LL.

Step 3 : 1 -> 2 -> 3 -> 4(ptr2) -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 (ptr1)

So here you have!, the 6th node from the end pointed to by ptr2!

## Q3 : Write a C program to return the nth node from the end of a linked list.

```
mynode * nthNode(mynode *head, int n)
{
        mynode *ptr1,*ptr2;  int count;
        if(!head)   return(NULL);
        ptr1 = ptr2 = head;
        count = 0;
        while(count < n)  {
                count++;
                        if((ptr1->next) != NULL)
                                ptr1 =ptr1->next;
                else
                                return NULL;
        }
        while((ptr1->next) != NULL)
        {
                ptr1 = ptr1->next;
                ptr2=ptr2->next;
        }
        return(ptr2);
}
```

**Time Complexity :** O(n)

**Q4: Write a function to get the intersection point of two Linked lists (Y Shape).**

**Q4: Write a function to get the intersection point of two Linked lists (Y Shape).**

**Brute Force Approach :** Using 2 for loops.

Outer loop will be for each node of the 1st list

Inner loop will be for 2nd list.

**Q4: Write a function to get the intersection point of two Linked lists (Y Shape).**

**Solution 2:**

Find the length of both the linked list

Find the difference in length (d)

For the longer linked list, traverse d node and check the addresses with another linked list.

# Q4: Write a function to get the intersection point of two Linked lists (Y Shape).

```
int getIntesectionNode(struct node* head1, struct node* head2)
{
   int c1 = getCount(head1);
   int c2 = getCount(head2);
   int d;
    d = abs( c1 - c2);
   if(c1 > c2)
   {
       return getIntesectionNode(d, head1, head2);
   }
   else
   {
       return getIntesectionNode(d, head2, head1);
   }
}
```

## Q4: Write a function to get the intersection point of two Linked lists (Y Shape).

```
int getIntesectionNode(int d, struct node* head1, struct node* head2){
  int i;
  struct node* current1 = head1;
  struct node* current2 = head2;
  for(i = 0; i < d; i++)
  {
    if(current1 == NULL)
        return -1;
    current1 = current1->next;
  }
        while(current1 !=  NULL && current2 != NULL)
    {   // we are comparing the addresses not data
        if(current1 == current2)
                return current1->data;
        current1= current1->next;
        current2= current2->next;
    }
    return -1;
}
```

# Q4: Write a function to get the intersection point of two Linked lists (Y Shape).

```
int getCount(struct node* head)
{
    struct node* current = head;
    int count = 0;
    while (current != NULL)
    {
        count++;
        current = current->next;
    }
    return count;
}
```

**Time Complexity** : O(m+n)

**Space Complexity :** O(1)