

# Implementing stacks using Linked lists

Writing stack functions using singly linked  
list

# Stack implementation using linked list

- **Property of stack:** Last in first out
  - **Stack functions:** Push and pop
  - **Objective: *Create a linked list that behaves like a stack!!***
  - Let us examine the properties of stack once again and then evaluate whether a linked list can be a good choice for implementing stack....
1. ***Properties of Stack:*** It is a dynamic data structure where push and pop operations are frequently performed, and stack grows and shrinks with those operations
  2. ***Similarities with linked list:*** Linked list contains similar type of elements as stack contains, it is dynamic and has the facility to insert and delete items easily

# Stack implementation using linked list

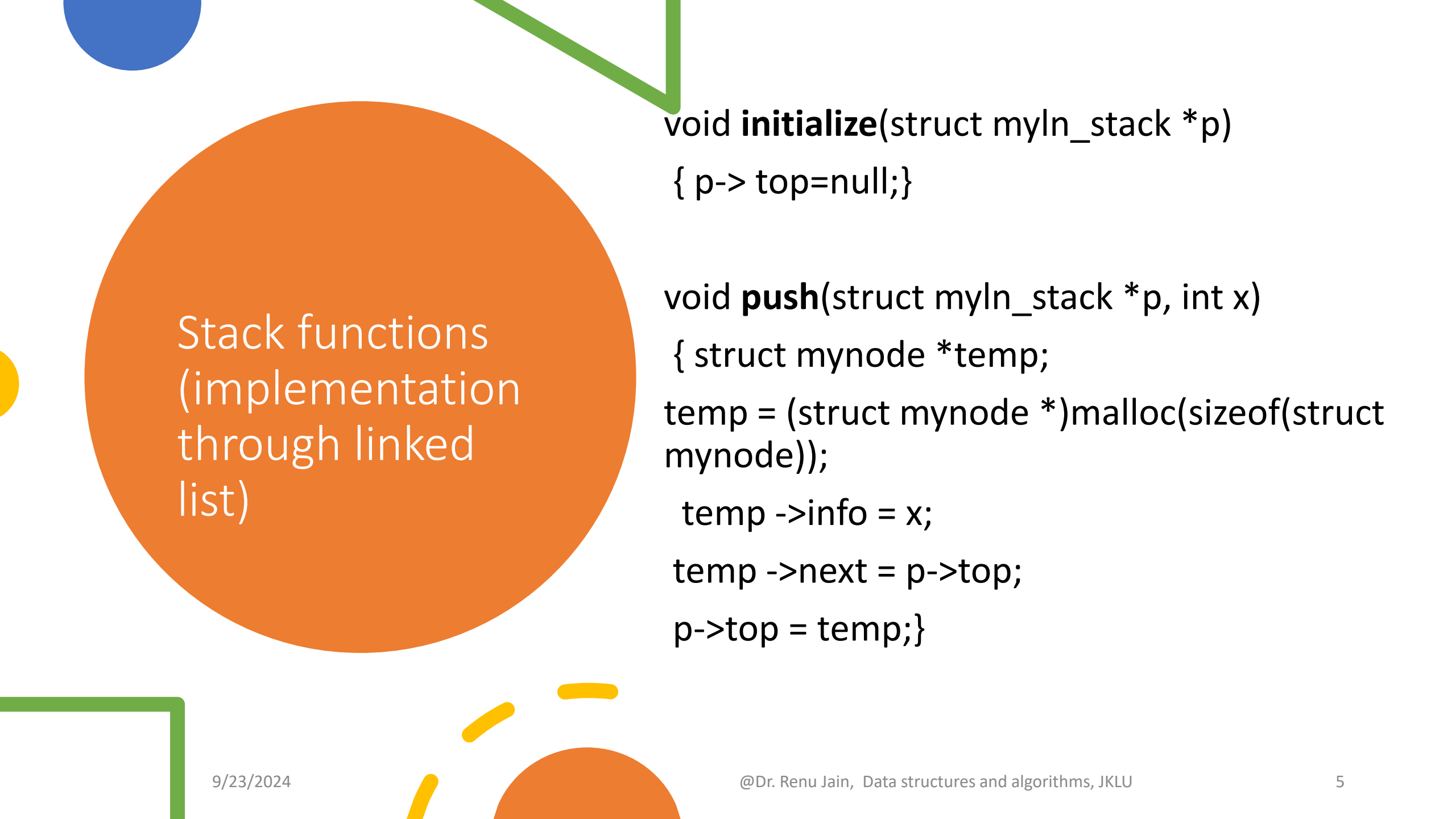
- **Imagine a linked list that we want to make it work as a stack...**
- Let us see whether it is possible or not!!!!
- 1. Stack must have a ***top pointer***
  - In linked list, ***start pointer*** can be treated as ***top pointer***
  - So, ***first element (the element pointed by start)*** of linked list is the top ***most element*** of stack
- 2. **Push**: Always insert a node before the first ***element*** if stack
- 3. **Pop**: Always delete the first element of linked list if stack

# Stack implementation using linked list

- Structure of stack: (if implemented as linked list)

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

```
struct myln_stack { struct mynode* top;}
```



## Stack functions (implementation through linked list)

```
void initialize(struct myln_stack *p)  
{ p-> top=null;}
```


```
void push(struct myln_stack *p, int x)  
{ struct mynode *temp;  
temp = (struct mynode *)malloc(sizeof(struct  
mynode));  
temp ->info = x;  
temp ->next = p->top;  
p->top = temp;}
```

# Stack implementation using linked list

```
int pop(struct myln_stack *p)
{ struct mynode temp;
  int x;
  if(p->top!= null)
  {x= p->top ->info ;
   temp = p->top;
   p->top = p->top ->next;
   free(temp);
   return x;} else return -100;
}
```



# Sample program



```
int main()
{ int x,y; struct myln_stack s1;
  initialize(&s1);
  scanf("%d\n", &x);
  Push(&s1,x);
  Y = pop(&s1);
  Printf("y= %d\n",y);
}
```

# Stack implementation (without using separate structure)

Note: It is possible that many books have this implementation but the previous implementation is more logical and secure.

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

//We just define top pointer for the stack as struct mynode * top;(not a global variable)

void initialize(struct mynode **p)
{ *p=null;}

void push(struct mynode **p, int x)
{ struct mynode temp;
  temp.info = x; temp.next = *p;  *p = &temp;} OR

void push(struct mynode **p, int x)
{ struct mynode *temp;
  temp = (struct mynode *)malloc(sizeof(struct mynode));
  temp ->info = x; temp ->next = *p; *p = temp;}
```



# Stack implementation (without using separate structure)

```
int pop(struct myln_stack **p)
{ struct mynode *temp; int x;
  if(*p != null)
  {x= *p ->info ;
   temp = *p;
   *p= *p->next;
   free(temp);
   return x;}
  else return -100;
}
```

# Cont..

```
int main()
{ int x,y; struct mynode *top;
  initialize(&top);
  scanf("%d\n", &x);
  push(&top,x);
  y = pop(&top);
  Printf("y= %d\n",y);
}
```

# Complexity....

# Queue implementation using linked list

- Property of queue: First in first out
  - Queue functions: Insert and Delete
  - Now we want that a linked list should behave like a queue
  - What makes us think that a linked list can be a good choice for implementing queue.....
1. Queue is a dynamic data structure where insertions and deletions are often performed, and it grows and shrinks with those operations
  2. Queue contains similar type of elements as linked list
  3. But queue has two ends!!! (linked list has only one end)

# Queue implementation using linked list

- Let us try whether we can implement queue as a linked list or not...
- Imagine a linked list as a Queue:
- Space is not a problem...dynamic allocation.
- **Start** can become the **front** of the queue
- So, in linked list, ***start pointer*** can be treated as a front pointer pointing to element “first in queue” and that should be deleted first
- The new element should be added at the end of list. How!! Keep a pointer ***last*** which points to last element of linked list (or queue)
- Is it easy or difficult to have another pointer....??

# Queue implementation using linked list

- Structure of Queue:

```
struct mynode {int data;  
                struct mynode * next;  
            }  
  
struct myln_queue{ struct mynode * front;  
                  struct mynode * rear;}}
```

# Queue implementation using linked list

```
void initialize(struct myln_queue *p)
{ p-> front=NULL;
  p-> rear=NULL;}
```

# Cont..

```
void insert(struct myln_queue *p, int x)
{
    struct mynode *temp;
    temp = (struct mynode* )malloc(sizeof(struct mynode));
    temp ->info = x; temp->next = null;
    if(p->rear == null)
    {
        p->rear= temp; p->front = temp;
    }
    else
    {
        p->rear->next = temp; p->rear = temp;
    }
}
```

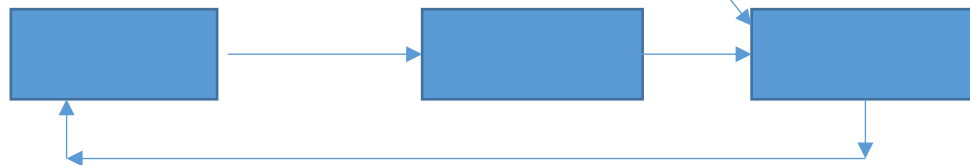


# Queue implementation using linked list

```
int delete(struct myln_queue *p)
{ struct mynode *temp; int x;
  if(p->rear == null) {printf("queue empty\n");}
else
{ x = p->front->info;
  temp = p->front;
  p->front = p->front->next;
  if((p->front) == null) p->rear= null;
  free(temp); return(x); }
```

# Circular linked list

- Let us make a small change to a simple linear linked list so that last node points to the first node instead of null.
- Such a list is called Circular linked list.
- So, a circular list does not have a natural first and last node, but we will establish first and last by convention
- Check advantages or disadvantages?
- We can do so by keeping a pointer to last node and with help of this we can find the address of first node also. ***last***



# stack and queue implementation using circular linked list

- Structure of stack:

```
struct mynode {int data;  
                struct mynode* next;  
            }  
  
struct cln_stack{ struct mynode*  cir_last;}  
struct cln_queue{ struct mynode*  cir_last;}
```

# Implementation of stack using circular linked list

```
struct mynode {int data;  
               struct mynode * next;  
               }  
struct cln_stack{ struct mynode* last;}
```

```
void initialize( struct cln_stack *p)  
{ p->last=null;}
```

// last pointer points to the last element of circular linked list. For stack implementation, **stack top pointer is last->next**

**Hence, in push, we will add before last->next and in pop, we will delete last-next**

# Cont..

```
void push(struct cln_stack *p, int x)
{ struct mynode *temp;
  temp = (struct mynode *)malloc(sizeof(struct mynode));
  temp ->info = x; temp ->next = null;
  if(p->last == null) {p->last =temp; temp->next = temp;}
  else {temp->next = p->last->next; p->last ->next = temp;}
```

# Implementation of stack using circular linked list, cont..

```
int pop(struct cln_stack *p)
{ struct mynode *temp; int x;
  if(p->last == null) { printf("can not pop\n"); return -1;}
  else { x = p->last ->next->info;
        if(p->last == p->last->next ) {free(p->last); p->last = null;}
        else { temp = p->last->next; p->last->next = temp->next; free(temp);}
        return x;
      }
```

# Implementation of queue using circular linked list

```
struct mynode {int data;  
                struct mynode * next;  
            }  
struct cln_queue{ struct mynode* last;}  
  
void initialize( struct cln_queue *q)  
{ q->last=null;}
```

# Cont..

```
void insert(struct cln_queue *q, int x)
{ struct mynode *temp;
  temp = (struct mynode *)malloc(sizeof(struct mynode));
  temp ->info = x; temp ->next = null;
  if(q->last == null) {q->last =temp; temp->next = temp;}
  else {temp->next = q->last->next; q->last->next = temp; q->last = temp;}
```



cont..

```
int delete(struct cln_queue *q)
{ struct mynode *temp; int x;
  if(q->last == null) { printf("can not delete\n"); return -1;}
  else { x = q->last ->next->info;
        if(q->last == q->last->next ) {free(q->last); q->last = null;}
        else { temp = q->last->next;
              q->last->next = temp->next; free(temp);
              return x;
            }
        }
}
```

# Array implementation of linked lists

- A list is an ordered collection of nodes where first node contains the address of next and so on..
- If a PL does not support dynamic memory allocation and we want to implement the concept of linked lists because deletion and addition in linked lists are simple
- `Struct node_type(int info; int next;)`
- `Struct node-type node[100]; (Assume start = 1){there are 4 elements in the list}`



# Doubly linked list

```
struct d_node{ int info;  
                struct d_node *left;  
                struct d_node *right;  
            };
```

Insert and delete in doubly linked list

# Practice problems(output problems)

```
void fun1(struct Node* head)
{
    if(head == NULL)
        return;

    fun1(head->next);
    printf("%d ", head->data);
}
```

fun1() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1.

## Cont...

```
void fun2(struct Node* head)
{
    if(head == NULL)
        return;
    printf("%d ", head->data);

    if(head->next != NULL )
        fun2(head->next->next);
    printf("%d ", head->data);
}
```

fun2() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then fun2() skips the last node. For Linked List 1->2->3->4->5, fun2() prints 1 3 5 5 3 1. For Linked List 1->2->3->4->5->6, fun2() prints 1 3 5 5 3 1.

# Some basic programming problems

1. Print the  $i$ th element of linked list
2. Print the middle element of the linked list
3. Delete the elements from linked list having value  $< 0$ .
4. Remove duplicate elements from a linked list
5. Merge two linked list
6. Break a linked list into two: 1 to  $i-1$ th element and  $i$ th element to last element
7. Delete a linked list
8. Store a very large integer into a linked list
9. Store a polynomial in a linked list

# Assignments

Write a program to add two polynomials. Store each polynomial in a linked list and generate a new linked list after adding the polynomials. Write a function to display the polynomial from linked list.

Addition of long positive integers using circular linked lists.