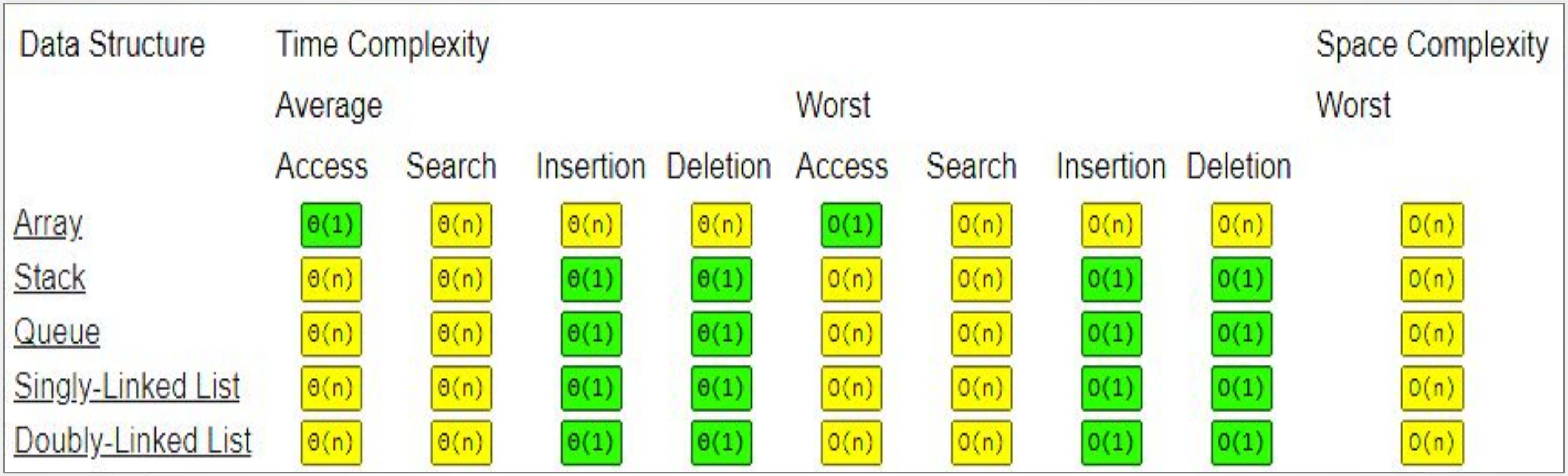
Stack and Queue Using LinkedList

Implement a stack using singly linked list

- Stack is LIFO/ FILO
- Stack requires O(n) space and its operations (Push and Pop) take O(1) time.
-

- Linked List is dynamic and does not require to follow LIFO or FIFO pattern
- Linked List requires O(n) space and its operations (Insert and Delete) take O(1) time.

Time and Space Requirements



Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

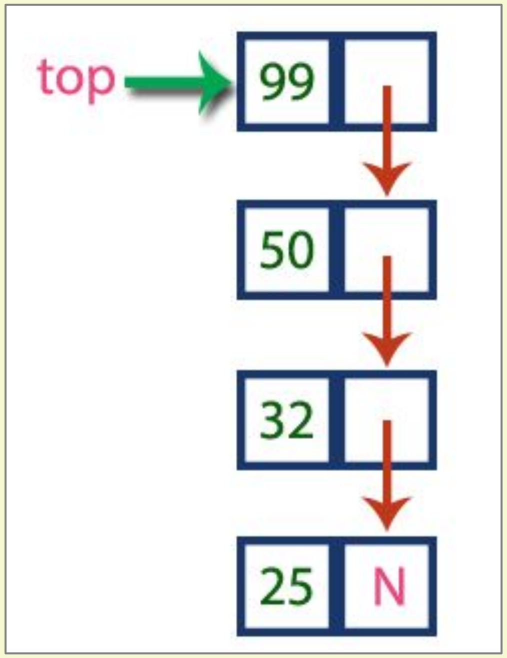
Stack Implementation

- A stack can be easily implemented through the linked list.
- In stack Implementation, a stack contains a top pointer. which is “head” of the stack where pushing and popping items happens at the head of the list.
- first node have null in link field and second node link have first node address in link field and so on and last node address in “top” pointer.
- The main advantage of using linked list over an arrays is that it is possible to implements a stack that can shrink or grow as much as needed.
- In using array will put a restriction to the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocate. so overflow is not possible.

Stack Implementation

Major Stack Operations:

- Push() : Insert the element into linked list nothing but which is the top node of Stack.
- Pop() : Return top element from the Stack and move the top pointer to the second node of linked list or Stack.
- display(): Print all element of Stack.



Initialize

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define a Node pointer 'top' and set it to NULL.
Step 4 - Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

```
#include<stdio.h>
#include<conio.h>

struct Node
{
   int data;
   struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();
```

```
void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Stack using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
       .
        .
        .
        .
   }
}
```

push(value) - Insert into the Stack

Step 1 - Create a newNode with given value.
Step 2 - Check whether stack is Empty (top == NULL)
Step 3 - If it is Empty, then set newNode → next = NULL.
Step 4 - If it is Not Empty, then set newNode → next = top.
Step 5 - Finally, set top = newNode.

```
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
   else
      newNode->next = top;
   top = newNode;
   printf("\nInsertion is Success!!!\n");
}
```

pop() - Delete from a Stack

Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display "Stack is Empty!!! UNDERFLOW!!!" and terminate the function
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
Step 4 - Then set 'top = top → next'.
Step 5 - Finally, delete 'temp'. (free(temp)).

```
void pop()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      printf("\nDeleted element: %d", temp->data);
      top = temp->next;
      free(temp);
   }
}
```

display() - Display Stack

Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display 'No Element In Stack!!!' and terminate the function.
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).
Step 5 - Finally! Display 'temp → data ---> NULL'.

```
void display()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
      }
      printf("%d--->NULL",temp->data);
   }
}
```

Implement a Queue using singly linked list

- Queue is FIFO/ LILO
- Queue requires O(n) space and its operations (Enqueue and Dequeue) take O(1) time.

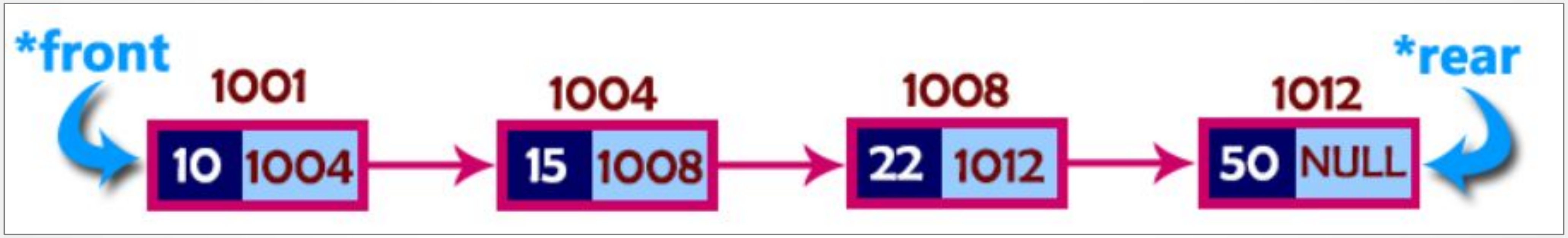
- Linked List is dynamic and does not require to follow LIFO or FIFO pattern
- Linked List requires O(n) space and its operations (Insert and Delete) take O(1) time.

Queue

Major Operations:

enQueue() This operation adds a new node after rear and moves rear to the next node.

deQueue() This operation removes the front node and moves front to the next node.



In linked list implementation of a queue, the last inserted node is always pointed by 'rear' and the first node is always pointed by 'front'.

In above example, the last inserted node is 50 and it is pointed by 'rear' and the first inserted node is 10 and it is pointed by 'front'. The order of elements inserted is 10, 15, 22 and 50.

Initialize

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.
Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

```
#include<stdio.h>
#include<conio.h>
struct Node
{
   int data;
   struct Node *next;
}*front = NULL,*rear =
NULL;
void insert(int);
void delete();
void display();
```

```
void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Queue Implementation using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
        .
        .
   }
}
```

enQueue(value) – Insert into the Queue

Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.
Step 2 - Check whether queue is Empty (rear == NULL)
Step 3 - If it is Empty then, set front = newNode and rear = newNode.
Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

```
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
      front = rear = newNode;
   else{
      rear -> next = newNode;
      rear = newNode;
   }
   printf("\nInsertion is Success!!!\n");
}
```

deQueue() - Delete from Queue

Step 1 - Check whether queue is Empty (front == NULL).
Step 2 - If it is Empty, then display "UNDERFLOW!!!" and terminate from the function
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

```
void delete()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      front = front -> next;
      printf("\nDeleted element: %d\n", temp->data);
      free(temp);
   }
}
```

display() - Display Queue

Step 1 - Check whether queue is Empty (front == NULL).
Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
Step 5 - Finally! Display 'temp → data ---> NULL'.

```
void display()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
      }
      printf("%d--->NULL\n",temp->data);
   }
}
```

THANK YOU