**A**

**Project Report**

on

“STACKS WITH DOUBLY LINKED LIST”

**Submitted by**

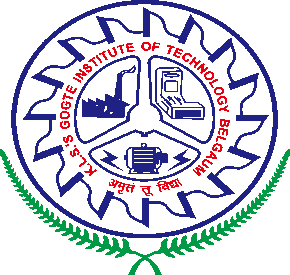
by

#### Sumeet P Bafna (USN: 2GI19CS157)

#### Venkatesh Dhongadi (USN: 2GI19CS175)

#### Vivek Jadhav (USN: 2GI19CS185)

#### Vivek Valagadde (USN: 2GI19CS186)



Department of Computer Science Engineering

KLS Gogte Institute of Technology,

Udyambag, Belgaum - 590008, Karnataka, India

**Theory :**

**Stack :** A stack is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.

The basic concept can be illustrated by thinking of your data set as a stack of plates or books where you can only take the top item off the stack in order to remove things from it. This structure is used all throughout programming.

The basic implementation of a stack is also called a Last in First Out structure; however there are different variations of stack implementations.

There are basically three operations that can be performed on stacks. They are:

* Push - it specifies adding an element to the Stack. If we try to insert an element when the Stack is full, then it is said to be Stack Overflow condition
* Pop - it specifies removing an element from the Stack. Elements are always removed from top of Stack. If we try to perform pop operation on an empty Stack, then it is said to be Stack Underflow condition.
* Peek - it will show the element on the top of Stack(without removing it)



A stack may be implemented to have a bounded capacity. If the stack is full and does

not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack. A pop either reveals previously concealed items or results in an empty stack, but, if the stack is empty, it goes into underflow state, which means no items are present in stack to be removed. A stack is a *restricted data structure*, because only a small number of operations are performed on it. The nature of the pop and push operations also means that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition. Therefore, the lower elements are

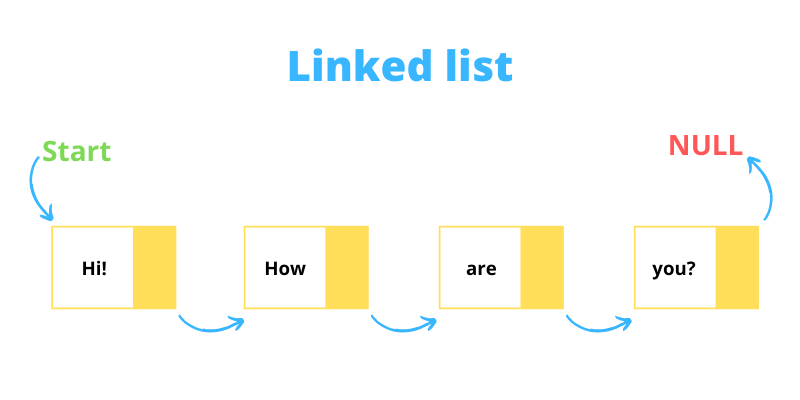
those that have been on the stack the longest.

**LINKED LIST :** It is the linear collection of data elements called nodes that are stored in different memory location connected by pointers. A linked list data structure is a set of records linked together by references.

The records are often called *nodes*. Each node points to another node. The references are often called *links* or *pointers*. From here on, the words *node* and *pointer* will be used for these concepts. In linked data structures, pointers are only dereference or compared for equality.

Thus, linked data structures are different than arrays, which require adding and

subtracting pointers.



**Advantages of linked list:-**

Dynamic data structure that can grow an string. Efficient memory utilization (exact amount of data storage). Insertion deletion & pupation are easy & efficient.Data stored in RAM but not sequential .

**Disadvantages of linked list:-**

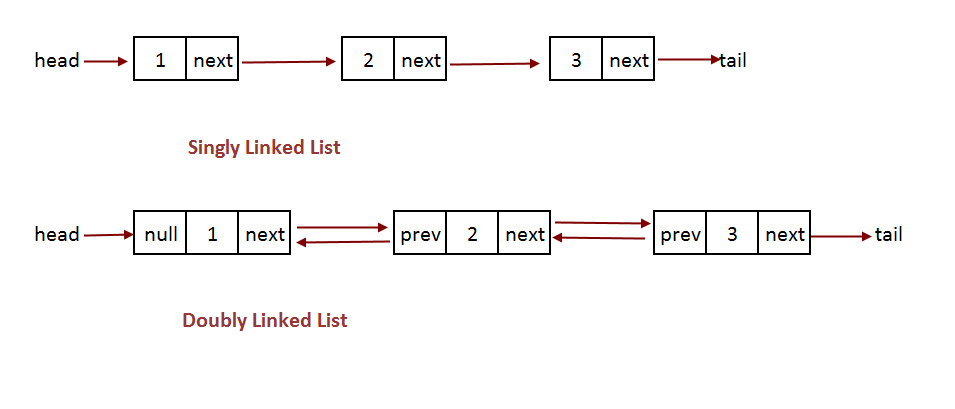
More memory space is needed if no. of filed are more. Logical & physical ordering of node are different. Searching is solve . Difficult to program because pointer manipulation is required.

**Types of linked list:-**

Linear linked list or one way linked list or single list. Double linked list or two way

linked list are two way linked list. Circular linked list is two types i.e. (1) Single

circular list (2) Double circular list.



**Linear linked list:-** It is a one way collection of nodes where the linear order is maintained by pointers. Nodes are not in sequence, each node implemented in comp. by a self referential structure. Each node is divided in two parts. First part contain the information of the element (INFO).

Second part is linked field contains the add. Of next node in the list (LINK)

field or next pointer filed .In c linked list is created using structured pointer

and Mallow (Allocation of memory). The structure of a node is strict node.

{

Into info;

Strict node\*link;

};

The new node is created and addresses of the new node is assigned to stack

has start.

{START =(Strict node\*) mallow(size of (strict node))}

**Doubly Linked List :** Unlike a Singly Linked List, in a Doubly Linked List, there is an extra pointer, the previous pointer along with the next pointer as shown below. The previous pointer points to the previous node in the list and the next pointer points to the next node in the list. So, in a doubly linked list, a node contains a reference both to its previous node and to its next node.  
  
**Advantages and Disadvantages of Doubly Linked List over Singly Linked List :**

Traversal can be done in both directions (from the start node to the end node as well as from the end node to the start node) in a Doubly Linked list. But this is not possible in a Singly Linked List and it can only be traversed only in one direction.  
Deletion and insertion operations are easy to implement in a Doubly LL than a Singly LL. For example, in a singly linked list, to delete a node, the pointer to the previous node is needed for which the list is to be traversed. In a Doubly LL, we just need to know the pointer of the node to be deleted.  
Memory has to be allocated for both the next and previous pointers in a node. Hence, the occupation of memory is higher in Doubly LL.  
Both the pointers will have to be modified if any kind of operation is performed like insertion, deletion, etc in case of Doubly LL

**Applications :**

* The browser cache which allows you to hit the BACK buttons or navigate through previous pages.
* Applications that have a Most Recently Used (MRU) lists.
* A stack, hash table, and binary tree can be implemented using a doubly linked list.
* Undo features in publishing or editing applications like Photoshop and Word.
* A great way to represent a deck of cards in a game where removing items from anywhere in the deck is essential.

**Program :**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

struct stack {

int data;

struct stack \*next;

struct stack \*prev;

};

typedef struct stack \*S;

/\*Push elements in to stack \*/

void push(S \*stack)

{

int e;

S temp;

temp=malloc(sizeof(struct stack));

if(!temp){

printf("Can't allocate memory\n");

return;

}

printf("Enter element to push\n");

scanf("%d", &e);

temp->data=e;

if(\*stack == NULL){

temp->next = temp->prev = NULL;

\*stack=temp;

} else {

temp->next = \*stack;

(\*stack)->prev = temp;

temp->prev = NULL;

\*stack=temp;

}

}

/\* Display elements in the stack \*/

void display(S s){

S temp;

temp=s;

printf("Elements are ...\n");

while(temp) {

printf("%d\n", temp->data);

temp=temp->next;

}

}

/\* Pop out elements in stack \*/

void pop(S \*stack)

{

S del;

if(\*stack == NULL){

printf("Stack is Empty ..\n");

return;

}

printf("Deleted .. %d\n", (\*stack)->data);

del = \*stack;

\*stack = (\*stack)->next;

/\* free memory once poped \*/

free(del);

if((\*stack)){

(\*stack)->prev = NULL;

}

}

/\* main program that help you to simulate stack operations\*/

int main()

{

int ch;

S s=NULL;

while(1){

printf("1 Push\n2 Pop\n3 Display\n4 exit\n");

printf("Enter you choice\n");

scanf("%d", &ch);

switch(ch) {

case 1: push(&s); break;

case 2: pop(&s); break;

case 3: display(s); break;

case 4: exit(0);

default: printf("invalid choice\n");

}

// getchar();

// getchar();

}

return 0;

}

**Conclusion :**