Experiment 9 **Date:04/10/2025**

Stack Using Array

Aim:

Program to implement stack operations using arrays.

<u>Algorithm:</u>

Main()

1. Start
2. Initialize top = -1
3. Repeat

    Display menu (1.Push 2.Pop 3.Display 4.Exit)

    Read choice

    If choice = 1

        Read value

        Call PUSH(value)

    Else if choice = 2

        Call POP()

    Else if choice = 3

        Call DISPLAY()

    Else if choice = 4

        Exit loop

    Else

        Print "Invalid Choice"

  Until choice = 4

4. Stop

Void push()

1. If top == SIZE - 1

    Print "Stack Overflow"

    Return

2. top = top + 1

3. stack[top] = value

4. Print "Value pushed"

5. End

Void pop()

1. If top == -1

    Print "Stack Underflow"

    Return

2. Print stack[top] as popped element

3. top = top - 1

4. End

Void display()

1. If top == -1

    Print "Stack is Empty"

    Return

2. Print "Stack elements:"

3. For i = top down to 0

    Print stack[i]

4. End

Program

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 5
void push(int value);
void pop();
void display();
int stack[SIZE];
int top=-1;
int main()
{
int choice,value;
do
{
```

```
printf("Stack operations:\n1.push\n2.pop\n3.display\n4.exit\n");
printf("Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
        printf("Enter value to push\n");
        scanf("%d",&value);
        push(value);
        break;
case 2:
        pop();
        break;
case 3:
        display();
        break;
case 4:
        exit(0);
        break;
default:
        printf("Invalid choice\n");
        break;
}
}while(choice!=4);
}
void push(int value)
{
if(top==SIZE-1)
{
printf("Stack overflow\n");
}
else
{
top++;
stack[top]=value;
printf("%d pushed into stack\n",value);
}
}
void pop()
{
if(top==-1)
{
printf("stack underflow");
}
else
{
printf("%d popped from stack\n",stack[top]);
```

```
top--;
}
}
void display()
{
if(top==-1)
{
printf("Stack is empty");
}
else
{
printf("Stack elements are:\n");
for(int i=top;i>=0;i--)
{
printf("%d\n",stack[i]);
}
}
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc stk_arry.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:1
Enter value to push
10
10 pushed into stack

Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:1
Enter value to push
20
20 pushed into stack

Stack operations:
1.push
```

```
2.pop
3.display
4.exit
Enter your choice:1
Enter value to push
30
30 pushed into stack

Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:3
Stack elements are:
30
20
10

Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:2
30 popped from stack

Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:3
Stack elements are:
20
10

Stack operations:
1.push
2.pop
3.display
4.exit
Enter your choice:4
```

Experiment 10 **Date:04/10/2025**

Queue Using Array

Aim:

Program to implement queue operations using arrays

<u>Algorithm:</u>

Main()

Start

Read n (size of queue)

Repeat

1. Display menu
2. Read choice
3. If choice = 1 → call enqueue(n)
4. Else if choice = 2 → call dequeue()
5. Else if choice = 3 → call traversal()
6. Else if choice = 4 → Exit

End loop

Stop

Void enqueue

If rear = size − 1

Print "Queue Full"

Stop

Read item

If front = -1 AND rear = -1

Set front = rear = 0

Else

Set rear = rear + 1

Insert item into a[rear]

End

Void dequeue

If front = -1 AND rear = -1

Print "Queue Empty"

Else

Store item = a[front]

If front = rear

Set front = rear = -1

Else

Set front = front + 1

Print "Item Removed"

End

Void traversal

If rear < 0

Print "Queue Empty"

Else

For i = front to rear

Print a[i]

End

Program

```
#include <stdio.h>
#include <stdlib.h>
void enqueue(int size);
void dequeue();
void traversal();
int a[20],front=-1,rear=-1;
void main(){
int n,ch;
printf("Enter the limit : ");
scanf("%d",&n);
```

```
do{
printf("1.Enqueue \n2.Dequeue \n3.Traversal \n4.Exit \nEnter your Choice: ");
scanf("%d",&ch);
switch(ch){
case 1: enqueue(n);
break;
case 2: dequeue();
break;
case 3: traversal();
break;
case 4:exit(0);
break;
default : printf("Enter a valid option!! ");
}
}while(ch!=4);
}

void enqueue(int size){
int item;
if(rear==size-1){
printf("Queue size is full!\n");
}
printf("Enter the element to insert :\n");
scanf("%d",&item);
if (front==-1 && rear==-1){
front=rear=0;
}
else{
rear = rear+1;
```

```
}
a[rear] = item;
}

void dequeue(){
if(front==-1 && rear== -1)
{
printf("Queue is empty\n");
}
else{
int item;
item = a[front];
if(front==rear){
front=rear=-1;
}
else{
front=front +1;
}
printf("ITEM REMOVED :\n");
}
}

void traversal(){
if(rear<0){
printf("Queue underflow!\n");
}
else{
int i;
for(i=front;i<=rear;i++){
```

```
printf("%d\t",a[i]);
}
}
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc queuee.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
Enter the limit : 5
1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 1
Enter the element to insert :
10

1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 1
Enter the element to insert :
20

1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 1
```

Enter the element to insert :

30

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 1

Enter the element to insert :

40

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 1

Enter the element to insert :

50

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 3

10 20 30 40 50

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 2

ITEM REMOVED

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 3

20 30 40 50

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 4

Experiment 11 **Date:06/10/2025**

Circular Queue

Aim:

Program to implement circular queue using array.

Algorithm:

Main()

Start

Read size of queue

If size <= 0 OR size > 20 → set size = 10

Repeat

Display menu

Read choice

If choice = 1 → call enqueue()

If choice = 2 → call dequeue()

If choice = 3 → call display()

If choice = 4 → Exit

End loop

Stop

Void enqueue()

If count == size

Print "Queue is full"

Stop

Read item

Insert item at a[rear]

Update rear = (rear + 1) % size

Increase count = count + 1

If count == 0

Print "Queue is empty"

Stop

Set item = a[front]

Update front = (front + 1) % size

Decrease count = count - 1

Print removed item

EndPrint inserted item

End

Void dequeue()

If count == 0

Print "Queue is empty"

Stop

Set item = a[front]

Update front = (front + 1) % size

Decrease count = count - 1

Print removed item

End

Void display()

If count == 0

Print "Queue is empty"

Stop

Set current_index = front

Repeat for i = 0 to count - 1

Print a[current_index]

Update current_index = (current_index + 1) % size

End loop

End

Program

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

```
#include<stdlib.h>
void enqueue();
void dequeue();
void display();
int a[20], front = 0, rear = 0, count = 0, size;
int main()
{
   int ch;
   printf("Enter the limit (max 20): ");
   scanf("%d", &size);
   if (size <= 0 || size > 20) {
      printf("Invalid size entered. Defaulting to 10.\n");
      size = 10;
   }
   do {
      printf("\n1. Enqueue \n2. Dequeue \n3. Display\n4. Exit \nEnter your Choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            enqueue();
            break;
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
```

```
        default:
            printf("Enter a valid option!! \n");
        }
    } while (ch != 4);
    return 0;
}
void enqueue() {
    int item;
    if (count == size) {
        printf("Queue size is full!\n");
    } else {
        printf("Enter the element to insert: ");
        scanf("%d", &item);
        a[rear] = item;
        rear = (rear + 1) % size;
        count = count + 1;
        printf("Inserted %d\n", item);
    }
}
void dequeue() {
    if (count == 0) {
        printf("Queue is empty\n");
    } else {
        int item;
        item = a[front];
        front = (front + 1) % size;
        count = count - 1;
        printf("ITEM REMOVED: %d\n", item);
    }
```

```
}
void display() {
   if (count == 0) {
      printf("Queue is empty!\n");
   } else {
      printf("Queue elements: ");
      int i;
      int current_index = front;
      for (i = 0; i < count; i++)
       {
         printf("%d ", a[current_index]);
         current_index = (current_index + 1) % size;
      }
      printf("\n");
   }
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc cir_q.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
Enter the limit (max 20): 5
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your Choice: 1
Enter the element to insert: 10
Inserted 10

1. Enqueue
```

2. Dequeue

3. Display

4. Exit

Enter your Choice: 1

Enter the element to insert: 20

Inserted 20

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 1

Enter the element to insert: 30

Inserted 30

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 1

Enter the element to insert: 40

Inserted 40

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 3

Queue elements: 10 20 30 40

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 2

ITEM REMOVED: 10

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 3

Queue elements: 20 30 40

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your Choice: 4

Experiment 12 **Date:10/10/2025**

Singly LinkedList Insertion

Aim:

To implement the following operations on a singly linked list

i. Creation
ii. Insert a new node at front
iii. Insert an element after a particular node
iv. Insert a new node at end
v. Searching
vi. Traversal.

Algorithm:

Main()

Start

Set head = NULL

Call createnode(head)

Repeat

Display menu

Read user choice

If choice = 1 → head = insertatfront(head)

If choice = 2 → head = insertatlast(head)

If choice = 3 → head = insertatpos(head)

If choice = 4 → call traverse(head)

If choice = 5 → read value and call valuesearch(head, value)

If choice = 6 → Exit

End loop

Stop

createnode(head)

Read number of nodes n

Repeat for i = 1 to n

Create a new node

Read data value

Set newnode→data = value

Set newnode→next = NULL

If head = NULL

   head = newnode

Else

   Traverse to last node

   last→next = newnode

Return head

Void insertatfront(head)

Create a new node

Read value

Set newnode→data = value

Set newnode→next = head

Set head = newnode

Return head

Void insertatlast(head)

Create a new node

Read value

Set newnode→data = value

Set newnode→next = NULL

If head = NULL

   head = newnode

Else

   Traverse to last node

   last→next = newnode

Return head

Void insertatpos(head)

Read value and position

Create a new node

Set newnode→data = value

If position = 1

newnode→next = head

head = newnode

Else

Traverse to node at position − 1

If position invalid → print error

Else

newnode→next = ptr→next

ptr→next = newnode

Return head

Void traverse(head)

If head = NULL

Print "List empty"

Else

Set temp = head

While temp ≠ NULL

Print temp→data

temp = temp→next

Stop

Void valuesearch(head, val)

Set temp = head, position = 1

While temp ≠ NULL

If temp→data = val

Print "Found at position"

Stop

Else

temp = temp→next

position = position + 1

If not found → Print "Value not found"

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node* createnode(struct node* head)
{
struct node *p;
int value,n;
printf("enter size \n");
scanf("%d",&n);
if(n <= 0) {
printf("List size must be greater than 0.\n");
return 0;
}
for(int i=1;i<=n;i++)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
```

```
temp->data=value;
temp->next=NULL;
if (head==NULL)
{
head=temp;
}
else
{
p=head;
while(p->next!=NULL)
{
p=p->next;
}
p->next=temp;
}
}
return head;
}
struct node* insertatfront(struct node* head);
struct node* insertatlast(struct node* head);
struct node* insertatpos(struct node* head);
void traverse(struct node* head);
void valuesearch(struct node* head,int val);
void main()
{
int ch,data,pos,val;
struct node* head = NULL;
printf("Creating a linked list:\n");
head=createnode(head);
```

```
do
{
printf("Linked List Operations\n1.Insert Node At Front\n2.Insert Node At Last\n3.Insert Node At ParticularPosition\n4.Traversal\n5.Searching\n6.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=insertatfront(head);
printf("value inserted at front\n");
break;
}
case 2:
{
head=insertatlast(head);
printf("value inserted at last\n");
break;
}
case 3:
{
head=insertatpos(head);
break;
}
case 4:
{
printf("traversing\n");
traverse(head);
break;
```

```
}
case 5:
{
printf("searching\n");
printf("enter value to search\n");
scanf("%d",&val);
valuesearch(head,val);
break;
}
case 6:exit(0);
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=6);
}

struct node* insertatfront(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
```

```
}
else
{
newnode->next=head;
head=newnode;
}
return head;
}
struct node* insertatlast(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
struct node *ptr;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
```

```
}
ptr->next=newnode;
return head;
}
struct node* insertatpos(struct node* head)
{
int value,pos;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
printf("enter position\n");
scanf("%d",&pos);
struct node *ptr;
int i;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{

if(pos==1)
{
newnode->next=head;
head=newnode;
printf("value inserted at position\n");
```

```
}
else
{
ptr=head;
for(i=1;i<pos-1&&ptr!=NULL;i++)
{
ptr=ptr->next;
}
if(ptr==NULL||ptr->next==NULL)
{
printf("\nPosition out of range\n");
}
else
{
newnode->next=ptr->next;
ptr->next=newnode;
printf("value inserted at position\n");
}
}
}
return head;
}
void traverse(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("list empty\n");
```

```
}
else
{
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}
void valuesearch(struct node* head,int val)
{
struct node *ptr;
ptr=head;
int flag=0,pos=1;
while(ptr!=NULL)
{
if(ptr->data==val)
{
flag=1;
printf("Item Present at position %d\n",pos);
break;
}
else
{
ptr=ptr->next;
pos=pos+1;
}
```

```
}
if(flag==0)
{
printf("Item Not Found\n");
}
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc sll_insert.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
Creating a linked list:
enter size
3
enter value to insert
10
enter value to insert
20
enter value to insert
30

Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Insert Node At ParticularPosition
4.Traversal
5.Searching
6.Exit
choose an operation:
1
enter value to insert
```

5

value inserted at front

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Insert Node At ParticularPosition

4.Traversal

5.Searching

6.Exit

choose an operation:

2

enter value to insert

40

value inserted at last

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Insert Node At ParticularPosition

4.Traversal

5.Searching

6.Exit

choose an operation:

3

enter value to insert

25

enter position

3

value inserted at position

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Insert Node At ParticularPosition

4.Traversal

5.Searching

6.Exit

choose an operation:

4

traversing

5->10->25->20->30->40->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Insert Node At ParticularPosition

4.Traversal

5.Searching

6.Exit

choose an operation:

5

searching

enter value to search

10

Item Present at position 2

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Insert Node At ParticularPosition

4.Traversal

5.Searching

6.Exit

choose an operation:

6

exit

Experiment 13 **Date:20/10/2025**

Singly LinkedList Deletion

Aim:

To implement the following operations on a singly linked list

i. Creation
ii. Deletion from beginning
iii. Deletion from the end
iv. Deletion from particular location
v. Traversal.

Algorithm:

Main()

Start.

Repeat the following forever:

Display menu:

1. Insert
2. Delete Beginning
3. Delete End
4. Delete by Value
5. Traverse
6. Exit

Read user's choice.

Use switch-case:

Case 1:

Read data.

Call insertEnd(data).

Case 2:

Call deleteBeginning().

Case 3:

Call deleteEnd().

Case 4:

Read key.

Call deleteByValue(key).

Case 5:

Call traverse().

Case 6:

Call freeList().

Print "Exiting".

Stop the program.

Default:

Print "Invalid Choice".

STEP 3: End.

Void insertEnd()

Create a new node.

Set new node's data = given value and next = NULL.

If head == NULL

head = new node

Else

Traverse list until last node

Set last node's next = new node

Stop.

Void deleteBeginning()

If head == NULL

Print "List empty" and stop.

temp = head

head = head->next

Free temp

Stop.

Void deleteEnd()

If head == NULL

Print "List empty" and stop.

If head->next == NULL

Delete head and set head = NULL

Stop.

Traverse list using two pointers:

prev = NULL

curr = head

While curr->next != NULL

prev = curr

curr = curr->next

Set prev->next = NULL

Free curr

Stop.

Void DeletebyValue()

If head == NULL → Stop.

If head->data == key

Delete head node

Stop.

Traverse list until found:

prev = current

current = current->next

If current == NULL

Key not found → Stop.

prev->next = current->next

Free current

Stop.

Void Traverse (Display List)

If head == NULL

Print "List is empty" and stop.

Set temp = head

While temp != NULL

Print temp->data

temp = temp->next

Stop.

Void FreetheList()

While head != NULL

temp = head

head = head->next

Free temp

Stop.

Program

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
Node* head = NULL;
Node* createNode(int data);
void insertEnd(int data);
```

```
void deleteBeginning();
void deleteEnd();
void deleteByValue(int key);
void traverse();
void freeList();
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Error: Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
void insertEnd(int data) {
   Node* newNode = createNode(data);
   if (head == NULL) {
      head = newNode;
      printf("Inserted %d (List was empty).\n", data);
      return;
   }
   Node* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
   printf("Inserted %d at the end.\n", data);
}
```

```
void traverse() {
    Node* temp = head;
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void deleteBeginning() {
    if (head == NULL) {
        printf("Cannot delete from the beginning. The list is empty.\n");
        return;
    }
    Node* temp = head;
    head = head->next;
    printf("Deleted from beginning: %d\n", temp->data);
    free(temp);
}
void deleteEnd() {
    if (head == NULL) {
        printf("Cannot delete from the end. The list is empty.\n");
        return;
    }
    if (head->next == NULL) {
```

```
        printf("Deleted from end: %d\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    Node* temp = head;
    Node* prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    printf("Deleted from end: %d\n", temp->data);
    free(temp);
}
void deleteByValue(int key) {
    Node* current = head;
    Node* prev = NULL;
    if (current != NULL && current->data == key) {
        head = current->next;
        printf("Deleted node with value %d (head).\n", key);
        free(current);
        return;
    }
    while (current != NULL && current->data != key) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
```

```
        printf("Value %d not found in the list. Cannot delete.\n", key);
        return;
    }
    prev->next = current->next;
    printf("Deleted node with value %d.\n", key);
    free(current);
}
void freeList() {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}
int main() {
    int choice, data, key;
    while (1) {
        printf("1. Insert (Creation)\n");
        printf("2. Delete from Beginning\n");
        printf("3. Delete from End\n");
        printf("4. Delete by Value (Particular Location)\n");
        printf("5. Traverse (Display List)\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
```

```
            scanf("%d", &data);
            insertEnd(data);
            break;
        case 2:
            deleteBeginning();
            break;
        case 3:
            deleteEnd();
            break;
        case 4:
            printf("Enter value to delete: ");
            scanf("%d", &key);
            deleteByValue(key);
            break;
        case 5:
            traverse();
            break;
        case 6:
            freeList();
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
   return 0;
}
```

Output

mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc sll_del.c

mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 1

Enter data to insert: 10

Inserted 10 (List was empty).

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 1

Enter data to insert: 20

Inserted 20 at the end.

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 1

Enter data to insert: 30

Inserted 30 at the end.

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 1

Enter data to insert: 40

Inserted 40 at the end.

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 2

Deleted from beginning: 10

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 3

Deleted from end: 40

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 4

Enter value to delete: 20

Deleted node with value 20 (head).

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 5

Linked List: 30 -> NULL

1. Insert (Creation)
2. Delete from Beginning
3. Delete from End
4. Delete by Value (Particular Location)
5. Traverse (Display List)
6. Exit

Enter your choice: 6

Exiting program.

Experiment 14 **Date:20/10/2025**

Stack Using Singly LinkedList

Aim:

To implement a menu driven program to perform following stack operations using linked list

i. push
ii. pop
iii. Traversal.

Algorithm:

Main()

Start.

Repeat forever:

Display menu:

1. Push
2. Pop
3. Traversal
4. Exit

Read choice.

If choice == 1, call push()

If choice == 2, call pop()

If choice == 3, call traversal()

If choice == 4, exit program

Else print "Invalid choice"

End.

Void push()

Read the value to be inserted.

Create a new node.

If memory allocation failed

    Print "Memory not allocated" and stop.

Set newNode->data = value.

Set newNode->next = top.

Set top = newNode.

Print "Pushed value".

Stop.

Void pop()

If top == NULL

    Print "Stack Underflow" and stop.

temp = top.

Print the popped element (temp->data).

Set top = top->next.

Free temp.

Stop.

If top == NULL

    Print "Stack empty" and stop.

Void traversal()

temp = top.

While temp != NULL

    Print temp->data

    Move temp to temp->next

Stop.

Program

```
#include <stdio.h>

#include <stdlib.h>

struct Node

{

   int data;

   struct Node *next;
```

```
};
struct Node *top = NULL;
void push();
void pop();
void traversal();
int main()
{
   int ch;
   while(1)
   {
      printf("1.push\n2.pop\n3.traversal\n4.exit\nEnter your choice: ");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1: push(); break;
         case 2: pop(); break;
         case 3: traversal(); break;
         case 4: exit(0);
         default: printf("Invalid\n");
      }
   }
   return 0;
}

void push()
{
   int value;
   struct Node *newNode;
   printf("Enter the value to insert: ");
```

```
    scanf("%d", &value);
    newNode = (struct Node*)malloc(sizeof(struct Node));
    if(newNode == NULL)
    {
        printf("Memory not allocated\n");
        return;
    }
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("Pushed %d\n", value);
}
void pop()
{
    struct Node *temp;
    if(top == NULL)
    {
        printf("Stack Underflow\n");
        return;
    }
    temp = top;
    printf("Popped %d\n", top->data);
    top = top->next;
    free(temp);
}
void traversal()
{
    struct Node *temp;
    if(top == NULL)
```

```
    {
        printf("Stack empty\n");
        return;
    }
    printf("Stack elements are: ");
    temp = top;
    while(temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc stk_ll.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
1.push
2.pop
3.traversal
4.exit
Enter your choice: 1
Enter the value to insert: 10
Pushed 10
1.push
2.pop
3.traversal
4.exit
Enter your choice: 1
Enter the value to insert: 20
```

Pushed 20

1.push

2.pop

3.traversal

4.exit

Enter your choice: 1

Enter the value to insert: 30

Pushed 30

1.push

2.pop

3.traversal

4.exit

Enter your choice: 40

Invalid

1.push

2.pop

3.traversal

4.exit

Enter your choice: 3

Stack elements are: 30 20 10

1.push

2.pop

3.traversal

4.exit

Enter your choice: 2

Popped 30

1.push

2.pop

3.traversal

4.exit

Enter your choice: 3

Stack elements are: 20 10

1.push

2.pop

3.traversal

4.exit

Enter your choice: 4

Experiment 15 **Date:03/11/2025**

Queue Using Singly LinkedList

Aim:

To implement a menu driven program to perform following queue operations using linked list

1. enqueue
2. dequeue
3. Traversal

Algorithm

Main()

Start.

Repeat:

Display menu:

1. Enqueue
2. Dequeue
3. Traversal
4. Exit

Read choice.

If choice = 1 → call enqueue()

Else if choice = 2 → call dequeue()

Else if choice = 3 → call traversal()

Else if choice = 4 → exit loop

Else print "Invalid option"

Stop.

Void enqueue()

Read the item to insert.

Create a new node.

If memory not allocated → print error and stop.

Set newNode->data = item

Set newNode->next = NULL

If rear == NULL (Queue is empty)

Set front = rear = newNode

Else

rear->next = newNode

rear = newNode

Stop.

Void dequeue()

If front == NULL

Print "Queue is empty" and stop.

temp = front

Print the removed element (temp->data)

Move front to next node:

front = front->next

If front == NULL

set rear = NULL

Free temp

Stop.

Void traversal()

If front == NULL

Print "Queue is empty" and stop.

temp = front

While temp != NULL

Print temp->data

Move temp to temp->next

Stop.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue() {
    int item;
    printf("Enter the element to insert:\n");
    scanf("%d", &item);
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    temp->data = item;
    temp->next = NULL;
    if (rear == NULL) {
        front = rear = temp;
    } else {
        rear->next = temp;
        rear = temp;
    }
}
```

```
void dequeue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct Node* temp = front;
    printf("ITEM REMOVED : %d\n", temp->data);
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    free(temp);
}

void traversal() {
    if (front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    struct Node* temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
int main() {
   int ch;
   do {
      printf("1.Enqueue \n2.Dequeue \n3.Traversal \n4.Exit \nEnter your Choice: ");
      scanf("%d", &ch);
      switch(ch) {
         case 1:
            enqueue();
            break;
         case 2:
            dequeue();
            break;
         case 3:
            traversal();
            break;
         case 4:
            break;
         default:
            printf("Enter a valid option!! \n");
      }
   } while (ch != 4);
   return 0;
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ gcc  ll_q.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
1.Enqueue
2.Dequeue
3.Traversal
```

4.Exit

Enter your Choice: 1

Enter the element to insert:

10

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 1

Enter the element to insert:

20

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 1

Enter the element to insert:

30

1.Enqueue

2.Dequeue

3.Traversal

4.Exit

Enter your Choice: 1

Enter the element to insert:

40

```
1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 3
Queue elements: 10 20       30      40

1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 2
ITEM REMOVED : 10

1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 3
Queue elements: 20 30       40

1.Enqueue
2.Dequeue
3.Traversal
4.Exit
Enter your Choice: 4
exit
```

Experiment 16 **Date:10/11/2025**

Doubly Linked List Operations-1

Aim:

To implement the following operations on a Doubly linked list.

i. Creation
ii. Count the number of nodes
iii. Searching
iv. Traversal

Algorithm

Main ()

Start.

Repeat

Display menu:

1. Creation
2. Searching
3. Display
4. Count nodes
5. Traverse
6. Exit

Read choice.

If choice = 1 → call create()

If choice = 2 → call search()

If choice = 3 → call display()

If choice = 4 → call counts()

If choice = 5 → call traverse()

If choice = 6 → exit loop

Else print "Invalid choice"

Stop.

Void create()

Read value for first node.

Create first node.

Set head = first.

Read number of additional nodes to create.

For each node:

Create new node.

Read value.

Set newnode->prev = last node.

Set last node->next = newnode.

Move last node pointer to newnode.

Stop.

Void search()

If head == NULL

Print "List empty" and stop.

STEP 2: Read value to search.

Set position = 0.

Traverse from head:

position++

If node->data == value

Print position and stop.

If value not found

Print "Element not found".

Stop.

Void display()

If head == NULL

Print "List empty" and stop.

Start from head.

While node != NULL:

Print node->data

Move to next node.

Stop.

Void counts()

If head == NULL

Print "List empty" and stop.

Set count = 0

Traverse through list:

count++

Print count.

Stop.

Void traverse()

If head == NULL

Print "List empty" and stop.

Set ptr = head.

While ptr != NULL:

Print ptr->data

Move ptr to ptr->next.

Stop.

Program

```
#include <stdio.h>
#include <stdlib.h>
void create(void);
void search(void);
void display(void);
void counts();
void traverse();
struct node {
```

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

struct node *head = NULL;

void main() {
    int ch;
    do {
        printf("\n\n1. Creation\n2. Searching\n3. Display\n4. Count of nodes\n5. Traverse\n6.
Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                create();
                break;
            case 2:
                search();
                break;
            case 3:
                display();
                break;
            case 4:
                counts();
                break;
            case 5:
                traverse();
                break;
```

```
        case 6:
            exit(0);
            break;
        default:
            printf("Invalid choice. Please try again.");
        }
    } while (ch != 6);
}

void create() {
    struct node *temp, *newnode, *first;
    int val, num;
    printf("\nEnter the value to be inserted: ");
    scanf("%d", &val);
    first = (struct node *)malloc(sizeof(struct node));
    first->data = val;
    first->next = NULL;
    first->prev = NULL;
    head = first;
    temp = head;
    printf("\nEnter the number of nodes to be created: ");
    scanf("%d", &num);
    for (int i = 1; i < num; i++) {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("\nEnter the value to be inserted: ");
        scanf("%d", &val);
        newnode->data = val;
        newnode->next = NULL;
        newnode->prev = temp;
```

```
        temp->next = newnode;
        temp = temp->next;
    }
    printf("\nDoubly Linked List created successfully");
}

void search() {
    struct node *ptr;
    int val;
    printf("\nEnter the value to be searched: ");
    scanf("%d", &val);
    int count = 0;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        int flag = 0, c1;
        ptr = head;
        while (ptr != NULL) {
            count++;
            if (ptr->data == val) {
                flag = 1;
                c1 = count;
                break;
            } else {
                ptr = ptr->next;
            }
        }
        if (flag) {
            printf("\nElement found at position %d", c1);
```

```
        } else {
            printf("\nElement not found");
        }
    }
}

void display() {
    struct node *ptr;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        ptr = head;
        while (ptr != NULL) {
            printf("%d <-> ", ptr->data);
            ptr = ptr->next;
        }
        printf("NULL");
    }
}

void counts() {
    struct node *ptr;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        int count = 0;
        ptr = head;
        while (ptr != NULL) {
            count += 1;
```

```
            ptr = ptr->next;
        }
        printf("\nThe number of nodes is %d ", count);
    }
}

void traverse() {
    struct node *ptr;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        printf("\nElements in the list (forward traversal): ");
        ptr = head;
        while (ptr != NULL) {
            printf("%d ", ptr->data);
            ptr = ptr->next;
        }
        printf("\n");
    }
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ dll.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS$ ./a.out
1. Creation
2. Searching
3. Display
4. Count of nodes
5. Traverse
6. Exit
```

Enter your choice: 1

Enter the value to be inserted: 10

Enter the number of nodes to be created: 4

Enter the value to be inserted: 20

Enter the value to be inserted: 30

Enter the value to be inserted: 40

Doubly Linked List created successfully

1. Creation
2. Searching
3. Display
4. Count of nodes
5. Traverse
6. Exit

Enter your choice: 2

Enter the value to be searched: 10

Element found at position 1

1. Creation
2. Searching
3. Display
4. Count of nodes
5. Traverse
6. Exit

Enter your choice: 3

10 <-> 20 <-> 30 <-> 40 <-> NULL

1. Creation
2. Searching

3. Display
4. Count of nodes
5. Traverse
6. Exit

Enter your choice: 4

The number of nodes is 4

1. Creation
2. Searching
3. Display
4. Count of nodes
5. Traverse
6. Exit

Enter your choice: 5

Elements in the list (forward traversal): 10 20 30 40

1. Creation
2. Searching
3. Display
4. Count of nodes
5. Traverse
6. Exit

Enter your choice: 6

Experiment 17 **Date:10/11/2025**

Doubly Linked List Operations-2

Aim:

To implement the following operations on a Doubly linked list.

i. Creation
ii. Insert a node at first position
iii. Insert a node at last
iv. Delete a node from the first position
v. Delete a node from last
vi. Traversal

Algorithm

Main()

Start.

Repeat

Display menu:

1. Creation
2. Insertion at beginning
3. Insertion at end
4. Display
5. Traverse
6. Delete from beginning
7. Delete from end
8. Exit

Read choice.

If choice = 1 → call create()

Else if choice = 2 → call insertbeg()

Else if choice = 3 → call insertend()

Else if choice = 4 → call display()

Else if choice = 5 → call traverse()

Else if choice = 6 → call deletefirst()

Else if choice = 7 → call deletelast()

Else if choice = 8 → exit program

Else print "Invalid choice"

Stop.

Void create()

Read value for first node.

Allocate memory for first node.

Set first->data = value, first->next = NULL, first->prev = NULL.

Set head = first and temp = head.

Read number of nodes to create (num).

For i = 1 to num-1:

Allocate newnode.

Read value.

Set newnode->data = value, newnode->next = NULL, newnode->prev = temp.

Set temp->next = newnode.

Set temp = newnode.

Print "Doubly Linked List created".

Stop.

Insertbeg ()

Read value to insert.

Allocate newnode.

Set newnode->data = value.

Set newnode->next = head and newnode->prev = NULL.

If head != NULL then head->prev = newnode.

Set head = newnode.

Print success message.

Stop

Void display()

If head == NULL then print "List empty" and stop.

Set ptr = head.

While ptr != NULL:

    Print ptr->data (e.g. "data <-> ")

    ptr = ptr->next

Print "NULL".

Stop.

Void traverse

If head == NULL then print "List empty" and stop.

Set ptr = head.

Print "Elements (forward): ".

While ptr != NULL:

    Print ptr->data

    ptr = ptr->next

Stop.

Void deletefirst()

If head == NULL then print "List empty" and stop.

temp = head.

head = head->next.

If head != NULL then head->prev = NULL.

Free temp.

Print success message.

Stop.

Void deletelast()

If head == NULL then print "List empty" and stop.

If head->next == NULL then

    free(head); head = NULL; print success; stop.

Set ptr = head.

While ptr->next != NULL do ptr = ptr->next.

ptr->prev->next = NULL.

Free ptr.

Print success message.

Stop.

Program

```
#include <stdio.h>

#include <stdlib.h>

void create(void);

void insertbeg(void);

void insertend(void);

void display(void);

void traverse();

void deletefirst(void);

void deletelast(void);

struct node {

    int data;

    struct node *next;

    struct node *prev;

};

struct node *head = NULL;

void main() {

int ch;

do {

printf("\n\n1. Creation\n2. Insertion at beginning\n3. Insertion at end\n4. Display\n5. Traverse\n6. Delete from beginning\n7. Delete from end\n8. Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch (ch) {

        case 1:
```

```
                create();
                break;
            case 2:
                insertbeg();
        break;
    case 3:
        insertend();
        break;
    case 4:
        display();
        break;
    case 5:
        traverse();
        break;
    case 6:
        deletefirst();
        break;
    case 7:
        deletelast();
        break;
    case 8:
        exit(0);
        break;
    default:
        printf("Invalid choice. Please try again.");
  }
} while (ch != 8);
}
void create() {
```

```
    struct node *temp, *newnode, *first;
    int val, num;
    printf("\nEnter the value to be inserted: ");
    scanf("%d", &val);
    first = (struct node *)malloc(sizeof(struct node));
    first->data = val;
    first->next = NULL;
    first->prev = NULL;
    head = first;
    temp = head;
    printf("\nEnter the number of nodes to be created: ");
    scanf("%d", &num);
    for (int i = 1; i < num; i++) {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("\nEnter the value to be inserted: ");
        scanf("%d", &val);
        newnode->data = val;
        newnode->next = NULL;
        newnode->prev = temp;
        temp->next = newnode;
        temp = temp->next;
    }
    printf("\nDoubly Linked List created successfully");
}

void insertbeg() {
    struct node *newnode;
    int val;
    newnode = (struct node *)malloc(sizeof(struct node));
```

```
    printf("\nEnter the value to be inserted: ");
    scanf("%d", &val);
    newnode->data = val;
    newnode->next = head;
    newnode->prev = NULL;
    if (head != NULL) {
        head->prev = newnode;
    }
    head = newnode;
    printf("\nSuccessful");
}

void insertend() {
    struct node *newnode;
    int val;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("\nEnter the value to be inserted: ");
    scanf("%d", &val);
    newnode->data = val;
    newnode->next = NULL;
    if (head == NULL) {
        newnode->prev = NULL;
        head = newnode;
    } else {
        struct node *ptr = head;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = newnode;
```

```
        newnode->prev = ptr;
    }
    printf("\nSuccessful");
}

void display() {
    struct node *ptr;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        ptr = head;
        while (ptr != NULL) {
            printf("%d <-> ", ptr->data);
            ptr = ptr->next;
        }
        printf("NULL");
    }
}

void traverse() {
    struct node *ptr;
    if (head == NULL) {
        printf("\nList empty!!!");
    } else {
        printf("\nElements in the list (forward traversal): ");
        ptr = head;
        while (ptr != NULL) {
            printf("%d ", ptr->data);
            ptr = ptr->next;
```

```
        }
        printf("\n");
    }
}


void deletefirst() {
    if (head == NULL) {
        printf("\nList empty, nothing to delete.");
        return;
    }
    struct node *temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
    printf("\nFirst node deleted successfully.");
}


void deletelast() {
    if (head == NULL) {
        printf("\nList empty, nothing to delete.");
        return;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
```

```
        printf("\nLast node deleted successfully (list is now empty).");
        return;
    }
    struct node *ptr = head;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->prev->next = NULL;
    free(ptr);
    printf("\nLast node deleted successfully.");
}
```

Output

```
mits@mits-Veriton-M200-H510:~/Faseeh/DS dll_del.c
mits@mits-Veriton-M200-H510:~/Faseeh/DS ./a.out
1. Creation
2. Insertion at beginning
3. Insertion at end
4. Display
5. Traverse
6. Delete from beginning
7. Delete from end
8. Exit
Enter your choice: 1
Enter the value to be inserted: 10
Enter the number of nodes to be created: 4
Enter the value to be inserted: 20
Enter the value to be inserted: 30
Enter the value to be inserted: 40
Doubly Linked List created successfully
```

1. Creation
2. Insertion at beginning
3. Insertion at end
4. Display
5. Traverse
6. Delete from beginning
7. Delete from end
8. Exit

Enter your choice: 2

Enter the value to be inserted: 5

Successful

1. Creation
2. Insertion at beginning
3. Insertion at end
4. Display
5. Traverse
6. Delete from beginning
7. Delete from end
8. Exit

Enter your choice: 3

Enter the value to be inserted: 50

Successful

1. Creation
2. Insertion at beginning
3. Insertion at end
4. Display

5. Traverse

6. Delete from beginning

7. Delete from end

8. Exit

Enter your choice: 5

Elements in the list (forward traversal): 5 10 20 30 40 50

1. Creation

2. Insertion at beginning

3. Insertion at end

4. Display

5. Traverse

6. Delete from beginning

7. Delete from end

8. Exit

Enter your choice: 6

First node deleted successfully.

1. Creation

2. Insertion at beginning

3. Insertion at end

4. Display

5. Traverse

6. Delete from beginning

7. Delete from end

8. Exit

Enter your choice: 7

Last node deleted successfully.

1. Creation

2. Insertion at beginning

3. Insertion at end

4. Display

5. Traverse

6. Delete from beginning

7. Delete from end

8. Exit

Enter your choice: 4

10 <-> 20 <-> 30 <-> 40 <-> NULL

1. Creation

2. Insertion at beginning

3. Insertion at end

4. Display

5. Traverse

6. Delete from beginning

7. Delete from end

8. Exit

Enter your choice: 8