Experiment 5 **Date:30/10/2024**

Array operations

Aim:

Write a program to implement a menu driven program to perform following array operation.

I. Insert an element to a particular location.
II. Delete an element in a particular location.
III. Traversal.

Algorithm:

main ()

1. Start
2. declare array arr and input array size
3. switch(choice)

   Case 1: call insertion(arr[], size)

   Case 2: call deletion(arr[], size)

   Case 3: call traversal(arr[], size)

   Default: invalid choice

4. stop

void insert (arr, size)

1. Start
2. Accept the index and element
3. for(i=size to index)
   a[i]=a[i-1]
4. a[index]=element
5. size++
6. exit

void delete (arr, size)

1. Start
2. Accept index want to delete
3. for(i=index to size-1)
   a[i]=a[i+1]
4. size=size-1
5. exit

void traversal (arr,size)

1. Start
2. for(1=0 to size)
3. print array element
4. exit

Program:

```
#include <stdio.h>
void displayArray(int arr[], int size) {
    if (size == 0) {
        printf("Array is empty.\n");
        return;
    }
    printf("Current Array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int insertElement(int arr[], int size) {
    int index, element;
    printf("Enter the index to insert the element (0 to %d): ", size);
    scanf("%d", &index);
    if (index < 0 || index > size) {
        printf("Invalid index.\n");
        return size;
    }
    printf("Enter the element to insert: ");
    scanf("%d", &element);
    for (int i = size; i > index; i--) {
        arr[i] = arr[i - 1];
    }
    arr[index] = element;
    size++;
    printf("Element '%d' inserted at index %d.\n", element, index);
    return size;
}
```

```
int deleteElement(int arr[], int size) {
    int index;
    if (size == 0) {
        printf("Array is empty. Cannot delete element.\n");
        return size;
    }
    printf("Enter the index to delete the element (0 to %d): ", size - 1);
    scanf("%d", &index);
    if (index < 0 || index >= size) {
        printf("Invalid index.\n");
        return size;
    }
    int removedElement = arr[index];
    for (int i = index; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    size--;
    printf("Element '%d' deleted from index %d.\n", removedElement, index);
    return size;
}
int main() {
    int arr[100];
    int size, i;
    int choice;

    printf("Enter the size of the array: ");
    scanf("%d", &size);
    printf("Enter the array elements:\n");
    for (i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
```

```
    do {
        printf("\nMenu:\n");
        printf("1. Insert Element\n");
        printf("2. Delete Element\n");
        printf("3. Traverse Array\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                size = insertElement(arr, size);
                break;
            case 2:
                size = deleteElement(arr, size);
                break;
            case 3:
                displayArray(arr, size);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);
    return 0;
}
```

Output

Enter the size of the array: 5

Enter the array elements:

1 2 3 4 5

Menu:

1. Insert Element

2. Delete Element

3. Traverse Array

4. Exit

Enter your choice: 1

Enter the index to insert the element (0 to 5): 3

Enter the element to insert: 9

Element '9' inserted at index 3.

Menu:

1. Insert Element

2. Delete Element

3. Traverse Array

4. Exit

Enter your choice: 3

Current Array: 1 2 3 9 4 5

Menu:

1. Insert Element

2. Delete Element

3. Traverse Array

4. Exit

Enter your choice: 2

Enter the index to delete the element (0 to 5): 3

Element '9' deleted from index 3.

Menu:

1. Insert Element

2. Delete Element

3. Traverse Array

4. Exit

Enter your choice: 3

Current Array: 1 2 3 4 5

Menu:

1. Insert Element

2. Delete Element

3. Traverse Array

4. Exit

Enter your choice: 4

Exiting the program.

Experiment 6 **Date:05/11/2024**

Array Sorting

Aim:

Program to sort an integer array

Algorithm:

1. Start.
2. Input the number of elements n.
3. Input the n elements into the array a[].
4. for i = 0 to n-1:

    for j = 0 to n-i-2:

        if a[j] > a[j+1]:

            temp = a[j]

            a[j] = a[j+1]

            a[j+1] = temp

4. Print the sorted array.
5. End.

Program:

```
#include<stdio.h>
void main()
{
int n,i,j,temp;
int a[10];
printf("Enter the limit:");
scanf("%d",&n);
printf("\n Enter Elements:");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
for(j=0;j<n-i-1;j++)
{
if( a[j]>a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
```

```
printf("sorted list are: \n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
}
```

Output

```
Enter the limit  : 9

Enter Elements: 22 4 21 98 43 12 78 23 45
 sorted list are:
 4 12 21 22 23 43 45 78 98
```

Experiment 7 **Date:05/11/2024**

Linear and Binary Searching

Aim:

Write Program to implement linear search and binary search

Algorithm:

Main()

1. Start
2. Read size n
3. Read n elements into array a[]
4. Read user ch

   If ch==1 Accept the element to search

   LinearSearch(a[], n, num) :

   If ch==2 Call sort(a[],n) to sort the array.

   Accept the element to search

   BinarySearch(a[], n, num) :

   If ch==3 exit the program:

5. End

Void LinearSearch(a[], n, num):

1. Initialize found = false.
2. For i = 0 to n-1:
   a. If a[i] == num:
      i. Print "Number found at index i".
      ii. Set found = true.
      iii. Break the loop.
3. If found is false:

   Print "Number not found".

4. exit

Void BinarySearch(a[], n, num):

1. Initialize low = 0, high = n-1.
2. While low <= high:

a. Set mid = (low + high) / 2.

b. If a[mid] == num:

i. Print "Number found at index mid".

ii. Return.

c. If a[mid] < num:

i. Set low = mid + 1.

d. If a[mid] > num:

i. Set high = mid - 1.

3. If the number is not found, print "Number not found".

4. exit

Void sort(a[], size):

1. for i = 0 to n-1:

for j = 0 to n-i-2:

if a[j] > a[j+1]:

temp = a[j]

a[j] = a[j+1]

a[j+1] = temp

2. print sorted array.

3. exit

Program:

```
#include <stdio.h>
void linear(int a[], int size, int num);
void binary(int a[], int size, int num);
void sort(int a[], int size);

int main() {
   int a[100], size, choice, num;

   printf("Enter size of the array : ");
   scanf("%d", &size);

   printf("Enter values to the array:\n");
   for (int i = 0; i < size; i++)
   {
      scanf("%d", &a[i]);
   }
```

```
        do {
            printf("\nSelect any one of the choices:\n");
            printf("1. Linear Search\n");
            printf("2. Binary Search\n");
            printf("3. Exit\n");
            printf("Enter your choice: ");
            scanf("%d", &choice);

            switch (choice) {
                case 1:
                    printf("Enter element to search (Linear Search): ");
                    scanf("%d", &num);
                    linear(a, size, num);
                    break;
                case 2:
                    sort(a, size);
                    printf("Enter element to search (Binary Search): ");
                    scanf("%d", &num);
                    binary(a, size, num);
                    break;
                case 3:
                  break;
                default:
                    printf("Invalid choice! Please try again.\n");
            }
        } while (choice != 3);

        return 0;
    }

void linear(int a[], int size, int num)
    {
        int found = 0;
        for (int i = 0; i < size; i++)
        {
            if (a[i] == num)
            {
                found = 1;
                printf("Number %d found at index %d (Linear Search)\n", num, i);
                break;
            }
        }
        if (!found)
        {
            printf("Number %d not found (Linear Search)\n", num);
        }
    }

void binary(int a[], int size, int num)
```

```
        {
            int low = 0, high = size - 1, mid;
            int found = 0;

            while (low <= high)
            {
                mid = (low + high) / 2;
                if (a[mid] == num)
                {
                    found = 1;
                    printf("Number %d found at index %d (Binary Search)\n", num, mid);
                    break;
                }
                else if (a[mid] < num)
                {
                    low = mid + 1;
                }
                else
                {
                    high = mid - 1;
                }
            }
            if (found==0)
            {
                printf("Number %d not found (Binary Search)\n", num);
            }
        }

void sort(int a[], int size)
        {
            for (int i = 0; i < size - 1; i++)
            {
                for (int j = 0; j < size - i - 1; j++)
                {
                    if (a[j] > a[j + 1])
                    {
                        int temp = a[j];
                        a[j] = a[j + 1];
                        a[j + 1] = temp;
                    }
                }
            }
        printf("sorted array : ");
        for(i=0;i<size;i++)
        {
        printf("%d\t",a[i]);
        }
    }
```

Output

```
Enter size of the array : 8
Enter values to the array:
18 34 27 88 45 8 32 55

Select any one of the choices:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 1
Enter element to search (Linear Search): 27
Number 27 found at index 2 (Linear Search)

Select any one of the choices:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 1
Enter element to search (Linear Search): 43
Number 43 not found (Linear Search)

Select any one of the choices:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 2
Sorted array:8 18 27 32 34 45 55 88
Enter element to search (Binary Search): 8
Number 8 found at index 0 (Binary Search)

Select any one of the choices:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 2
Sorted array:8 18 27 32 34 45 55 88
Enter element to search (Binary Search): 43
Number 43 not found (Binary Search)

Select any one of the choices:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 3
Exiting……
```

Experiment 8 **Date: 07/11/2024**

Matrix Operations

Aim:

To implement a menu driven program to perform the following matrix operations
Addition
Subtraction
Multiplication

Algorithm:

main()
1.Start
2.Declare a[20][20],b[20][20],m,n,p,q,i,j,ch.
3.Read the order of matrix1.
4.Read the order of matrix2.
5.Read the elements of matrix,matrix2 and display it.
6.Display choices.
7.Read option ch.
8.If ch==1 call add(a,b,m,n,p,q)
If ch==2 call sub(a,b,m,n,p,q)
If ch==3 call mul(a,b,n,p)
If ch==4 exit from the program
9.Repeat steps 6 and 7 while ch!=4
10.Stop

void add(a,b,m,n,p,q)
1. Start.
2. Declare c[20][20],i,j.
3. If (m==p && n==q)
For i=0 to m-1
For j=0 to n-1
do c[i][j] = a[i][j] + b[i][j] [end of loop]
[end of loop] Display the matrix.
Else
Print "Order of the matrices does not match" [end of if]
4. Exit.

void sub(a,b,m,n,p,q)
1. Start.
2. Declare c[20][20],i,j.
3. If (m==p && n==q)
For i=0 to m-1
For j=0 to n-1
Do c[i][j] = a[i][j] - b[i][j] [end of loop]
[end of loop] Display the matrix.

Else
Print “Order of the matrices does not match” [end of if]
4. Exit.

void mul(a,b,n,p)
1. Start.
2. Declare c[20][20],i,j,k.
If (n!=p)
Print “Matrix multiplication not possible” Else
For i=0 to m-1
  For j=0 to q-1
Set c[i][j]=0
    For k=0 to n-1
do c[i][j] = c[i][j] + a[i][k] * b[k][j] [end of loop]
[end of loop] [end of loop]
Display the matrix. [end of if]
Exit.

Program

```
 #include<stdio.h>
 #include<stdlib.h>
void add(int a[20][20], int b[20][20], int m, int n, int p, int q);
void sub(int a[20][20], int b[20][20], int m, int n, int p, int q);
void mul(int a[20][20], int b[20][20], int m, int n, int p, int q);

void main() {
int a[20][20], b[20][20], m, n, p, q, i, j, ch;
printf("Enter the order of matrix_1:");
scanf("%d%d", &m, &n);
printf("Enter the order of matrix_2:");
scanf("%d%d", &p, &q);
printf("Enter %d elements for matrix_1:", m * n);
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) { scanf("%d", &a[i][j]);
 }
}
printf("Enter %d elements for matrix_2:", p * q);
for (i = 0; i < p; i++) {
for (j = 0; j < q; j++) { scanf("%d", &b[i][j]);
         }
}
printf("Matrix_1:\n");
 for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
```

```
 printf("%d ", a[i][j]);
}
printf("\n");
}
printf("Matrix_2:\n");
 for (i = 0; i < p; i++) {
for (j = 0; j < q; j++) {
 printf("%d ", b[i][j]);
}
printf("\n");
}
do {
printf("\n.MENU");
printf("\n1.Addition\n2.Subtraction\n3.Multiplication\n4.Exit");    printf("\nEnter
your choice:");
scanf("%d", &ch); switch (ch) {
case 1:
  add(a, b, m, n, p, q); break;
case 2:
  sub(a, b, m, n, p, q); break;
case 3:
  mul(a, b, m, n, p, q); break;
case 4:
  exit(0); default:
printf("Invalid option. Please enter 1-4"); break;
}
   } while (ch > 0 && ch <= 4);
}

void add(int a[20][20], int b[20][20], int m, int n, int p, int q)
{
 int c[20][20], i, j;
if (m == p && n == q) {
   for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
    c[i][j] = a[i][j] + b[i][j];
}}
printf("\nMatrix after addition:\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
printf("%d ", c[i][j]);
}
printf("\n");
   }}
 else {
printf("Order of the matrices does not match");
```

```
}}

void sub(int a[20][20], int b[20][20], int m, int n, int p, int q) {
int c[20][20], i, j;
if (m == p && n == q) {
      for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) { c[i][j] = a[i][j] - b[i][j];
}}
printf("\nMatrix after subtraction:\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
 printf("%d ", c[i][j]);
}
printf("\n");
}
} else {
printf("Order of the matrices does not match");
   }}

void mul(int a[20][20], int b[20][20], int m, int n, int p, int q) {
int c[20][20], i, j, k;
if (n != p) {
printf("Matrix multiplication not possible. Columns of Matrix1 must match rows
of Matrix2.\n");
return;
}
   for (i = 0; i < m; i++) {
      for (j = 0; j < q; j++) {
         c[i][j] = 0;
         for (k = 0; k < n; k++) {
            c[i][j] += a[i][k] * b[k][j];
}
}
}
  printf("\nMatrix after multiplication:\n");
   for (i = 0; i < m; i++) {
      for (j = 0; j < q; j++) {
         printf("%d ", c[i][j]);
      }
      printf("\n");
   }
}
```

Output

```
Enter the order of matrix_1:2 2
Enter the order of matrix_2:2 2
Enter 4 elements for matrix_1:1 2 3 4
```

```
Enter 4 elements for matrix_2:5 6 7 8

Matrix_1:
1 2
3 4
Matrix_2:
5 6
7 8

…MENU…
1.Addition
2.Subtraction
3.Multiplication
4.Exit
Enter your choice:1
Matrix after addition:
6 8
10 12

…MENU…
1.Addition 2.Subtraction 3.Multiplication 4.Exit
Enter your choice:2
Matrix after subtraction:
4 4
4 4

…MENU…
1.Addition 2.Subtraction 3.Multiplication 4.Exit
Enter your choice:3
Matrix after multiplication:
19 22
43 50

…MENU…
1.Addition 2.Subtraction 3.Multiplication 4.Exit
Enter your choice:4
```

Experiment 9 **Date:07/11/2024**

Stack using Array

Aim:

Program to implement stack operations using arrays.

Algorithm:

1.Start

2.Declare max = 100, stack[100] and top = -1

3.read choice from user

if ch==1

call push()

if ch==2

call pop()

if ch==3

call display()

if ch==4

exit:

default: Invalid choice.

4.stop

void push()

1 Start

2 if top == max – 1

Print “Stack Overflow!”.

3 declare and read value to push

4 top = top + 1

5 stack[top] = value

6 print “value pushed onto stack.”

7 exit

void pop()

1 Start

2 if top == -1

Print “Stack Underflow!”.

3 declare value

4 value= stack[top]

5 print “value popped from stack..”

6 top = top - 1

7 exit

void display()

1 Start

2 if top == -1

Print “Stack is empty.”

3 for i = top to 0

4 print stack[i]

5 exit

Program:

```
#include <stdio.h>
int max = 100;
int stack[100];
int top = -1;
void push();
void pop();
void display();

int main() {
   int choice;
   do {
      printf("1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit \nEnter the
choice:");
      scanf("%d", &choice);

      switch (choice) {
          case 1:
            push();
            break;
         case 2:
```

```
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

void push() {
    if (top == max - 1) {
        printf("Stack Overflow! Cannot push more elements.\n");
    }
     else{
    int value;
    printf("Enter element to insert: ");
    scanf("%d", &value);
    top++;
    stack[top] = value;
    printf("%d pushed onto stack.\n", value);
}
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow! Nothing to pop.\n");
    }
    else{
    int value = stack[top];
    printf("%d popped from stack.\n", value);
    top--;
}
}

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

```
    }
}
}
```

Output

```
1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
 Enter the choice:1
Enter element to insert: 10
10 pushed onto stack.

1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
Enter the choice:1
Enter element to insert: 20
20 pushed onto stack.

1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
Enter the choice:1
Enter element to insert: 30
30 pushed onto stack.

1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
Enter the choice:1
Enter element to insert: 40
40 pushed onto stack.

1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
Enter the choice:2
40 popped from stack.

1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
Enter the choice:3
Stack elements:
30
20
10

Enter the choice:
1. Insertion (Push) 2. Deletion (Pop) 3. Display 4. Exit
4
Exiting...
```

Experiment 10 **Date:18/11/2024**

Queue using Arrays

Aim:

To implement a menu driven program to perform the following queue operations using array

a) Enqueue
b) Dequeue
c) Traverse

Algorithm:

Define queue size and global variables

1 Define queue size using #define SIZE 20
2 Declare a global array to represent queue, q[SIZE]
3 Declare and initialize front=-1, rear=-1

main()

1 Start
2 Declare ch.
3 Display choices.
4 Read option ch.

If ch==1 call enqueue()
If ch==2 call dequeue()
If ch==3 call traverse()
If ch==4 exit from the program

5 Repeat steps 3 and 4 while ch>0&&ch<=4
6 Stop

void enqueue()

1 Start.
2 Declare item.
3 If (rear == SIZE-1)

Print "Overflow"
Go to step 8

4 Read the element to be added to the queue.
5 If (front == -1)

Set front = 0

6 Set rear = rear+1
7 Set q[rear] = item
8 Exit.

void dequeue()

1 Start.
2 If (front == -1)

Print "Underflow"
Go to step 5.

3 Display dequeued element, q[front]
4 If (front == rear)
Set front = rear = -1
Else
Set front = front+1
5 Exit.

void traverse()

1 Start.
2 If (front == -1)
Print “Underflow”
Go to step 4.
3 Display elements present in the queue.
4 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
int front = -1, rear = -1, q[SIZE];
void enqueue();
void dequeue();
void traverse();
void main()
{
   int ch;
   do {
      printf("\nMENU");
      printf("\n1. Enqueue\n2. Dequeue\n3. Traverse\n4. Exit");
      printf("\n\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch)  {
      case 1:
         enqueue();
         break;
      case 2:
         dequeue();
         break;
      case 3:
         traverse();
         break;
      case 4:
         exit(0);
      default:
         printf("\nInvalid choice! Please enter a number between 1 and 4.");
      }   } while (ch >= 1 && ch <= 4);
}
```

```
void enqueue()
{
   int item;
   if (rear == SIZE - 1)    {
      printf("\nOverflow! The queue is full.");
      return;
   }
   printf("\nEnter the element to be added to the queue: ");
   scanf("%d", &item);
   if (front == -1) {
      front = 0;
   }
   rear++;
   q[rear] = item;
}
void dequeue()
{
   if (front == -1) {
      printf("\nUnderflow! The queue is empty.");
      return;
   }
   printf("\nDequeued element: %d", q[front]);
   if (front == rear)    {
      front = rear = -1;
   }
   else {
      front++;
   }}
void traverse()
{
   if (front == -1) {
      printf("\nThe queue is empty!");
      return;
   }
   printf("\nElements in the queue are:\n");
   for (int i = front; i <= rear; i++)    {
      printf("%d ", q[i]);
   }
   printf("\n");
}
```

Output

```
MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 2

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 4

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 2
Dequeued element: 2

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 2
Dequeued element: 3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
```

```
Enter your choice: 3
Elements in the queue are:
4

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 4
```

Experiment 11 **Date:18/11/2024**

Circular Queue using Arrays

Aim:

To implement a menu driven program to perform the following circular queue operations using array

a) Enqueue
b) Dequeue
c) Traverse

Algorithm:

Define queue size and global variables

1 Define queue size using #define SIZE 20
2 Declare a global array to represent queue, q[SIZE]
3 Declare and initialize front=-1, rear=-1

main()

1 Start
2 Declare ch.
3 Display choices.
4 Read option ch.

If ch==1 call enqueue()
If ch==2 call dequeue()
If ch==3 call traverse()
If ch==4 exit from the program

5 Repeat steps 3 and 4 while(1)
6 Stop

void enqueue()

1 Start.
2 Declare item.
3 If ((rear +1) % SIZE == front)

Print "Overflow"
Go to step 8

4 Read the element to be added to the queue.
5 If (front == -1)

Set front = 0
Set rear = 0

Else

Set rear = (rear+1) % SIZE

6 Set q[rear] = item
7 Exit.

void dequeue()

1 Start.
2 If (front == -1)

Print “Underflow”
Go to step 5.
3 Display dequeued element, q[front]
4 If (front == rear)
Set front = rear = -1
Else
Set front = (front+1) % SIZE
5 Exit.

void traverse()

1 Start.
2 If (front == -1)
Print “Underflow”
Go to step 4.
3 Display elements present in the queue.
4 Exit.

Program

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 20
int front = -1, rear = -1, q[SIZE];
void enqueue();
void dequeue();
void traverse();
void main()
{
   int ch;
   while(1){
      printf("\nMENU");
      printf("\n1.enQueue\n2.deQueue\n3.Traverse\n4.Exit");
      printf("\n\nEnter the choice: ");
      scanf("%d", &ch);
      switch (ch){
      case 1:
         enqueue();
         break;
      case 2:
         dequeue();
         break;
      case 3:
         traverse();
         break;
      case 4:
         exit(0);
      default:
         printf("\nInvalid choice!! Please enter 1-4");
```

```
    }}}
void enqueue()
{
 int item;
 if ((rear+1) % SIZE == front){
  printf("Overflow\n");  }
  else{
      printf("\nEnter the element to be added: ");
      scanf("%d", &item);
      if (front == -1){
          front = rear = 0;
      }
      else{
        rear = (rear + 1) % SIZE;
      }
      q[rear] = item;
  }}
void dequeue()
{
 if (front == -1){
      printf("\nUnderflow!!");   }
  else{
      printf("\nDequeued element: %d", q[front]);
      if (front == rear){
          front = rear = -1;
      }
      else {
        front = (front + 1) % SIZE;
      }  }  }
void traverse()
{
   int i;
   if (front == -1){
      printf("\nQueue is empty!!");
   }
   else{
      printf("\nElements present in the queue: \n");
      i = front;
      while(1){
        printf("%d ", q[i]);
        if (i == rear)
          break;
        i = (i + 1) % SIZE; }
      printf("\n");}}
```

Output

```
MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 1
Enter the element to be added: 6

MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 1
Enter the element to be added: 3

MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit

Enter the choice: 1
Enter the element to be added: 9
MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the queue:
6 3 9

MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 2
Dequeued element: 6

MENU
1.enQueue
2.deQueue
```

```
3.Traverse
4.Exit
Enter the choice: 2
Dequeued element: 3

MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the queue:
6

MENU
1.enQueue
2.deQueue
3.Traverse
4.Exit
Enter the choice: 4
```

Experiment 12 **Date: 20/11/2024**

Singly Linked List- Insertion

Aim:

To implement the following operations on a singly linked list

a) Creation
b) Insert a new node at front
c) Insert an element after a particular node
d) Insert a new node at end
e) Searching
f) Traversal

Algorithm:

Declare the structure node

struct node

1 Declare data, struct node *next

main()

1 Start
2 Declare ch, struct node *head.
3 Set head=NULL
4 To create first node call create(head)
5 Display choices.
6 Read option ch.

If ch==1 call insertbeg (head)
If ch==2 call insertlast (head)
If ch==3 call insertpos (head)
If ch==4 call search(head)
If ch==5 call traverse(head)
If ch==6 exit from the program

7 Repeat steps 5 and 6 while(1)
8 Stop

struct node* create(struct node *head)

1 Start.
2 Declare item.
3 Allocate memory for struct node *first.
4 Read the item that wanted to insert.
5 Set first->data=item
6 Set first->next=NULL
7 Set head=first
8 Return head
9 Exit.

struct node* insertbeg(struct node *head)

1 Start.
2 Declare item.
3 Allocate memory for struct node *temp.
4 If temp=NULL
Print “Memory insufficient”
Go to step 9
5 Read the item that wanted to insert.
6 Set temp->data=item
7 Set temp->next=head
8 Set head=temp
9 Return head
10 Exit.

struct node* insertlast(struct node *head)

1 Start.
2 Allocate memory for struct node *temp.
3 If temp=NULL
Print “Memory insufficient”
Go to step 8
4 Read the item that wanted to insert.
5 Set temp->data=item
6 Set temp->next=NULL
7 If head=NULL
Set head=temp
Else
Set *ptr=head
While ptr->next != NULL
Set ptr=ptr->next
Set ptr->next=temp
8 Return head
9 Exit.

struct node* insertpos(struct node *head)

1 Start.
2 Declare item, pos.
3 Allocate memory for struct node *temp.
4 If temp=NULL
Print “Memory insufficient”
Go to step 13
5 Read the item that wanted to insert.
6 Read the position where the item wanted to insert.
7 Set temp->data=items
8 If pos = 1
Set temp->next=head
Set head=temp

Print “Item is inserted at position 1”
Go to step 13
9 Set *ptr=head
10 For i=1 to i<pos-1 && ptr!=NULL
Set ptr=ptr->next
11 If ptr=NULL
Print “Position out of range”
Else
Set temp->next=ptr->next
Set ptr->next=temp
12 Return head
13 Exit.

void search (struct node *head)

1 Start.
2 If head=NULL
Print “List is empty”
Go to step 9
3 Declare item, flag=0, pos=1
4 Read the item that wanted to search.
5 Set *ptr=head
6 While ptr!=NULL
If ptr->data=item
Set flag=1
Print “Item present”
Set ptr=ptr->next
pos=pos+1
7 If flag=0
Print “Item not found”
8 Exit.

void traverse (struct node *head)

1 Start.
2 Declare strcut node *ptr
3 Set ptr=head
4 If ptr=NULL
Print “List is empty”
Go to step 6
5 While ptr!=NULL
Print ptr->data
Set ptr=ptr->next
Print “NULL”
6 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node* create(struct node *head);
struct node* insertbeg(struct node *head);
struct node* insertlast(struct node *head);
struct node* insertpos(struct node *head);
void search(struct node *head);
void traverse(struct node *head);
void main()
{
   struct node *head = NULL;
   int ch, item;
   printf("Creating first node:\n");
   head=create(head);
   while(1)
   {
      printf("\nMENU\n");
      printf("\n1.Insert a node at beginning\n2.Insert a node at end\n3.Insert a
node at a specific position\n4.Search\n5.Traverse\n6.Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch)
      {
        case 1:
                head = insertbeg(head);
                break;
        case 2:
                head = insertlast(head);
                break;
        case 3:
                head = insertpos(head);
                break;
        case 4:
                search(head);
                break;
        case 5:
                traverse(head);
                break;
        case 6:
```

```
                exit(0);
            default:
                printf("Invalid choice. Please enter 1-6\n");
        }
    }
}
struct node* create(struct node *head)
{
  int item;
  struct node *first=(struct node*)malloc(sizeof(struct node));
  printf("Enter the item you want to insert:");
  scanf("%d",&item);
  first->data=item;
  first->next=NULL;
  head=first;
  return head;
}
struct node* insertbeg(struct node *head)
{
   int item;
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   if (temp == NULL)
   {
      printf("Memory insufficient\n");
      return head;
   }
   printf("Enter the item you want to insert: ");
   scanf("%d", &item);
   temp->data = item;
   temp->next = head;
   head = temp;
   printf("One node is inserted at the beginning\n");
   return head;
}
struct node* insertlast(struct node *head)
{
   int item;
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   if (temp == NULL)
   {
      printf("Memory insufficient\n");
      return head;
   }
   printf("Enter the item you want to insert: ");
   scanf("%d", &item);
   temp->data = item;
```

```
    temp->next = NULL;
    if (head == NULL)
    {
        head = temp;
    }
    else
    {
        struct node *ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = temp;
    }
    printf("One node is inserted at the end\n");
    return head;
}
struct node* insertpos(struct node *head)
{
    int item, pos;
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    if (temp == NULL)
    {
        printf("Memory insufficient\n");
        return head;
    }
    printf("Enter the item you want to insert: ");
    scanf("%d", &item);
    printf("Enter the position you want to insert: ");
    scanf("%d", &pos);
    temp->data = item;
    if (pos == 1)
    {
        temp->next = head;
        head = temp;
        printf("Item is inserted at position 1\n");
        return head;
    }
    struct node *ptr = head;
    for (int i = 1; i < pos - 1 && ptr != NULL; i++)
    {
        ptr = ptr->next;
    }
    if (ptr == NULL)
    {
        printf("Position out of range\n");
```

```
    }
    else
    {
        temp->next = ptr->next;
        ptr->next = temp;
        printf("Item is inserted at position %d\n", pos);
    }
    return head;
}
void search(struct node *head)
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    int item, flag = 0, pos = 1;
    printf("Enter the element you want to search: ");
    scanf("%d", &item);
    struct node *ptr = head;
    while (ptr != NULL) {
        if (ptr->data == item)
        {
            flag = 1;
            printf("Item present at position %d\n", pos);
            break;
        }
        ptr = ptr->next;
        pos++;
    }
    if (flag == 0) {
        printf("Item not found\n");
    }
}
void traverse(struct node *head)
{
    struct node *ptr = head;
    if (ptr == NULL)
    {
        printf("List is empty\n");
        return;
    }
    while (ptr != NULL)
     {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
```

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

Output

```
Creating first node:
Enter the item you want to insert:1

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 5
1 -> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 1
Enter the item you want to insert: 2
One node is inserted at the beginning

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 5
2 -> 1 -> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
```

```
Enter your choice: 2
Enter the item you want to insert: 5
One node is inserted at the end

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 5
2 -> 1 -> 5 -> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 3
Enter the item you want to insert: 3
Enter the position you want to insert: 2
Item is inserted at position 2

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 5
2 -> 3 -> 1 -> 5 -> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 4
Enter the element you want to search: 1
Item present at position 3
```

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Insert a node at a specific position
4.Search
5.Traverse
6.Exit
Enter your choice: 6

Experiment 13 **Date: 02/12/2024**

Singly Linked List- Deletion

Aim:

To implement the following operations on a singly linked list

a) Creation
b) Deletion from beginning
c) Deletion from end
d) Deletion from particular location
e) Traversal

Algorithm:

Declare the structure node

struct node

1 Declare data, struct node *next

main()

1 Start
2 Declare ch, struct node *head.
3 Set head=NULL
4 To create a list of nodes call create(head)
5 Display choices.
6 Read option ch.
    If ch==1 call deletebeg (head)
    If ch==2 call deletelast (head)
    If ch==3 call deletepos (head)
    If ch==4 call traverse(head)
    If ch==5 exit from the program
7 Repeat steps 5 and 6 while(1)
8 Stop

struct node* create(struct node *head)

1 Start.
2 Declare item, n, i, struct node *temp, *first.
3 Read the number of nodes that wanted to create.
4 If n<=0
    Print “Number of nodes should be greater than zero”
    Go to step 6
5 For i=1 to i<=n
    Read the item for node
    Allocate memory for *temp
    If temp=NULL
        Print “Memory insufficient”
        Go to step 7
    Set temp->data=item
    Set temp->next=NULL
    Set first->next=temp
    Set first=temp

6 Return head
7 Exit.

struct node* deletebeg(struct node *head)

1 Start.
2 Allocate memory for struct node *ptr.
3 If head=NULL
  Print “Memory insufficient”
  Go to step 7
4 Set ptr=head
5 Set head=head->next
6 Free(ptr)
7 Return head
8 Exit.

struct node* deletelast(struct node *head)

1 Start.
2 If head=NULL
  Print “Memory insufficient”
  Go to step 4
3 If head->next=NULL
  Set head=NULL
  Free(head)
  Else
  Set *ptr=head
  While ptr->next != NULL
  Set ptr1=ptr
  Set ptr=ptr->next
  Set ptr1->next=NULL
  Free(ptr)
4 Return head
5 Exit.

struct node* deletepos(struct node *head)

1 Start.
2 Declare key.
3 If head=NULL
  Print “Memory insufficient”
  Go to step 10
4 Read the key that wanted to delete.
5 If head->data = key
  head=head->next
  go to step 10
6 Set *ptr1=head
7 Set *ptr=head->next
8 While ptr!=NULL
  If ptr->data=key
  Set ptr1->next=ptr->next
  free(ptr)

```
            go to step 10
          Else
            Set ptr1=ptr
            Set ptr=ptr->next
    9   If ptr=NULL
          Print “Node with key does not exist”
    10  Return head
    11  Exit.
```

void traverse (struct node *head)

```
    1   Start.
    2   Set *ptr=head
    3   If ptr=NULL
          Print “List is empty”
          Go to step 5
    4   While ptr!=NULL
           Print ptr->data
           Set ptr=ptr->next
        Print “NULL”
    5   Exit.
```

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node* create(struct node *head);
struct node* deletebeg(struct node *head);
struct node* deletelast(struct node *head);
struct node* deletepos(struct node *head);
void traverse(struct node *head);
void main()
{
   struct node *head = NULL;
   int ch, ch1,item;
   printf("Creating a linked list:\n");
   head=create(head);
   while(1)
   {
      printf("\nMENU\n");
      printf("1.Delete a node from the beginning\n2.Delete a node from the
end\n3.Delete a node from a specific position\n4.Traverse\n5.Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch)
```

```
        {
          case 1:
                    head = deletebeg(head);
                    break;
          case 2:
                    head = deletelast(head);
                    break;
          case 3:
                    head = deletepos(head);
                    break;
          case 4:
                    traverse(head);
                    break;
          case 5:
          exit(0);
          default:
          printf("Invalid choice. Please enter 1-5\n");
        }
    }
}
struct node* create(struct node *head)
{
    int n, item, i;
    struct node *temp, *first;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
       printf("Number of nodes should be greater than zero.\n");
       return head;
    }
    for (i = 1; i <= n; i++)
    {
       printf("Enter the item for node %d: ", i);
       scanf("%d", &item);
       temp = (struct node*)malloc(sizeof(struct node));
       if (temp == NULL) {
          printf("Memory insufficient\n");
          break;
       }
       temp->data = item;
       temp->next = NULL;
       first->next = temp;
       first = temp;
    }
    return head;
}
struct node* deletebeg(struct node *head)
{
    struct node *ptr = (struct node *)malloc(sizeof(struct node));
```

```
    if (head == NULL)
    {
       printf("Memory insufficient\n");
       return head;
    }
    ptr = head;
    head = head->next;
    free(ptr);
    printf("One node is deleted from the beginning\n");
    return head;
}
struct node* deletelast(struct node *head)
{
    if (head == NULL)
    {
       printf("Memory insufficient\n");
       return head;
    }
    if (head->next == NULL)
    {
       head = NULL;
       free(head);
    }
    else
    {
       struct node *ptr = head;
       struct node *ptr1;
       while (ptr->next != NULL)
       {
          ptr1 = ptr;
          ptr = ptr->next;
       }
       ptr1->next = NULL;
       free(ptr);

    }
    printf("One node is deleted from the end\n");
    return head;
}
struct node* deletepos(struct node *head)
{
    int key;
    if (head == NULL)
    {
       printf("Memory insufficient\n");
       return head;
    }
    printf("Enter the key  you want to delete: ");
    scanf("%d", &key);
```

```
    if (head->data == key)
    {
       head = head->next;
       return head;
    }
    struct node *ptr1 = head;
    struct node *ptr = head->next;
    while (ptr != NULL)
    {
     if (ptr->data == key)
     {
       ptr1->next = ptr->next;
       free(ptr);
       printf("Node is deleted\n");
       return head;
     }
     else
     {
       ptr1 = ptr;
       ptr = ptr->next;
     }
    }
    if (ptr == NULL)
    {
     printf("Node with key does not exist\n");
    }
    return head;
}
void traverse(struct node *head)
{
    struct node *ptr = head;
    if (ptr == NULL)
    {
       printf("List is empty\n");
       return;
    }
    while (ptr != NULL)
     {
       printf("%d -> ", ptr->data);
       ptr = ptr->next;
    }
    printf("NULL\n");
}
```

Output

```
Creating a linked list:
Enter the number of nodes you want to create: 5
```

```
Enter the item for node 1: 1
Enter the item for node 2: 2
Enter the item for node 3: 3
Enter the item for node 4: 4
Enter the item for node 5: 5

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
Enter your choice: 1
One node is deleted from the beginning

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
Enter your choice: 4
2 -> 3 -> 4 -> 5 -> NULL

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
Enter your choice: 3
Enter the key  you want to delete: 4
Node is deleted

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
Enter your choice: 4
2 -> 3 -> 5 -> NULL

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
```

```
Enter your choice: 2
One node is deleted from the end

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
Enter your choice: 4
2 -> 3 -> NULL

MENU
1.Delete a node from the beginning
2.Delete a node from the end
3.Delete a node from a specific position
4.Traverse
5.Exit
```

Experiment 14 **Date: 02/12/2024**

Stack using Singly Linked List

Aim:

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

a) Push
b) Pop
c) Traverse

Algorithm:

Declare the structure node and global variables

struct node

1 Declare data, struct node *next
2 Initialize struct node *top=NULL

main()

1 Start
2 Declare ch.
3 Display choices.
4 Read option ch.
    If ch==1 call push()
    If ch==2 call pop()
    If ch==3 call traverse()
    If ch==4 exit from the program
5 Repeat steps 3 and 4 while(1)
6 Stop

void push()

1 Start.
2 Declare item.
3 Read the element to be added onto the stack.
4 Allocate memory for *temp.
5 If (!temp)
    Print “Overflow”
    Go to step 9
6 Set temp->data=item
7 Set temp->next=top
8 Set top=temp
9 Exit.

void pop()

1 Start.
2 If (top == NULL)
    Print “Underflow”
    Go to step 7
3 Set *temp=top
4 Print “Popped element”
5 Set top=top->next

6 Free(temp)
7 Exit.

void traverse()

1 Start.
2 If (top == NULL)
  Print “Underflow”
  Go to step 5
3 Set *temp=top
4 Display elements present in the stack.
5 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node* next;
};
struct node* top = NULL;
void push();
void pop();
void traverse();
void main() {
   int ch;
   while(1) {
      printf("\nMENU");
      printf("\n1.Push the element\n2.Pop the element\n3.Traverse\n4.Exit");
      printf("\n\nEnter the choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            push();
            break;
         case 2:
            pop();
            break;
         case 3:
            traverse();
            break;
         case 4:
            exit(0);
         default:
            printf("\nInvalid choice!! Please enter 1-4");
      }
   }
}
void push() {
```

```
    int item;
    printf("\nEnter the element to be added onto the stack: ");
    scanf("%d", &item);

    struct node* temp = (struct node*)malloc(sizeof(struct node));
    if (!temp) {
       printf("\nOverflow");
       return;
    }
    temp->data = item;
    temp->next = top;
    top = temp;
}
void pop() {
    if (top == NULL) {
       printf("\nUnderflow!!");
       return;
    }
    struct node* temp = top;
    printf("\nPopped element: %d", top->data);
    top = top->next;
    free(temp);
}
void traverse() {
    if (top == NULL) {
       printf("\nUnderflow!!");
       return;
    }
    struct node* temp = top;
    printf("\nElements present in the stack: \n");
    while (temp != NULL) {
       printf("%d\n", temp->data);
       temp = temp->next;
    }
}
```

Output

```
MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 1
Enter the element to be added onto the stack: 1

MENU
```

```
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the stack:
1

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 1
Enter the element to be added onto the stack: 2

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the stack:
2
1

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 1
Enter the element to be added onto the stack: 3

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the stack:
3
2
1

MENU
1.Push the element
2.Pop the element
3.Traverse
```

```
4.Exit
Enter the choice: 2
Popped element: 3

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 3
Elements present in the stack:
2
1

MENU
1.Push the element
2.Pop the element
3.Traverse
4.Exit
Enter the choice: 4
```

Experiment 15 **Date: 02/12/2024**

Queue using Singly Linked List

Aim:

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

d) Enqueue
e) Dequeue
f) Traverse

Algorithm:

Declare the structure node and global variables

struct node

1 Declare data, struct node *next
2 Initialize struct node *front=NULL
3 Initialize struct node *rear=NULL

main()

1 Start
2 Declare ch.
3 Display choices.
4 Read option ch.
    If ch==1 call enqueue()
    If ch==2 call dequeue()
    If ch==3 call traverse()
    If ch==4 exit from the program
5 Repeat steps 3 and 4 while(1)
6 Stop

void enqueue()

1 Start.
2 Declare item.
3 Read the element to be added to the queue.
4 Allocate memory for *temp.
5 If (!temp)
    Print “Overflow”
    Go to step 9
6 Set temp->data=item
7 Set temp->next=NULL
8 If (rear=NULL)
    Set front=rear=temp
  Else
    Set rear->next=temp
    Set rear=temp
9 Exit.

void dequeue()

1 Start.
2 If (front =NULL)
    Print “Underflow”
    Go to step 8

3 Set *temp=front
4 Set front=front->next
5 If (front=NULL)
Set rear=NULL
6 Display dequeued element.
7 Free(temp)
8 Exit.

void traverse()

1 Start.
2 If (front =NULL)
Print “Underflow”
Go to step 5
3 Set *temp=front
4 Display elements present in the queue.
5 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node* next;
};
void enqueue();
void dequeue();
void traverse();
struct node* front = NULL;
struct node* rear = NULL;
void main() {
   int ch;
   while(1) {
      printf("\nMENU");
      printf("\n1. Enqueue\n2. Dequeue\n3. Traverse\n4. Exit");
      printf("\n\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            enqueue();
            break;
         case 2:
            dequeue();
            break;
         case 3:
            traverse();
            break;
         case 4:
            exit(0);
```

```
            default:
                printf("\nInvalid choice! Please enter a number between 1 and 4.");
        }
    }
}
void enqueue() {
    int item;
    printf("\nEnter the element to be added to the queue: ");
    scanf("%d", &item);
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("\nOverflow");
        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("\nUnderflow");
        return;
    }
    struct node* temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    printf("\nDequeued element: %d", temp->data);
    free(temp);
}
void traverse() {
    if (front == NULL) {
        printf("\nThe queue is empty!");
        return;
    }
    struct node* temp = front;
    printf("\nElements in the queue are:\n");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
```

```
}
```

Output

```
MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 1

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 2

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 1
Enter the element to be added to the queue: 3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 3
Elements in the queue are:
1 2 3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit

Enter your choice: 2
Dequeued element: 1

MENU
1. Enqueue
2. Dequeue
```

```
3. Traverse
4. Exit
Enter your choice: 3
Elements in the queue are:
2 3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 2
Dequeued element: 2

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 3
Elements in the queue are:
3

MENU
1. Enqueue
2. Dequeue
3. Traverse
4. Exit
Enter your choice: 4
```

Experiment 16 **Date: 04/12/2024**

Doubly Linked List- Simple Operations

Aim:

To implement the following operations on a doubly linked list

a) Creation
b) Count the number of nodes
c) Searching
d) Traversal

Algorithm:

Declare the structure node

struct node

1 Declare data, struct node *next, *prev

main()

1 Start
2 Declare ch, struct node *head.
3 Set head=NULL
4 To create a list of nodes call create(head)
5 Display choices.
6 Read option ch.
    If ch==1 call search(head)
    If ch==2 call count(head)
    If ch==3 call traverse(head)
    If ch==4 exit from the program
7 Repeat steps 1 and 4 while(1)
8 Stop

struct node* create(struct node *head)

1 Start.
2 Declare item, n, i, struct node *temp, *first.
3 Read the number of nodes that wanted to create.
4 If n<=0
    Print “Number of nodes should be greater than zero”
    Go to step 6
5 For i=1 to i<=n
    Read the item for node
    Allocate memory for *temp
    If temp=NULL
        Print “Memory insufficient”
        Go to step 7
    Set temp->data=item
    Set temp->next=NULL
    Set temp->prev=first
    Set first->next=temp
    Set first=temp

6 Return head
7 Exit.

void search (struct node *head)

1 Start.
2 If head=NULL
  Print “List is empty”
  Go to step 9
3 Declare item, flag=0, pos=1
4 Read the item that wanted to search.
5 Set *ptr=head
6 While ptr!=NULL
  If ptr->data=item
  Set flag=1
  Print “Item present”
  Go to step 9
  Set ptr=ptr->next
  pos=pos+1
7 If flag=0
  Print “Item not found”
8 Exit.

void count (struct node *head)

1 Start.
2 Declare count, struct node *ptr
3 Set count=0
4 Set ptr=head
5 If ptr=NULL
  Print “List is empty”
  Go to step
6 While ptr!=NULL
  count=count+1
  Set ptr=ptr->next
7 Display the count of nodes
8 Exit.

void traverse (struct node *head)

1 Start.
2 Declare struct node *ptr
3 Set ptr=head
4 If ptr=NULL
  Print “List is empty”
  Go to step 6
5 While ptr!=NULL
  Print ptr<->data
  Set ptr=ptr->next
  Print “NULL”
6 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *prev;
   int data;
   struct node *next;
};
struct node* create(struct node *head);
void search(struct node *head);
void count(struct node *head);
void traverse(struct node *head);
void main()
{
   struct node *head = NULL;
   int ch, item;
   printf("Creating a linked list:\n");
   head=create(head);
   while(1){
      printf("\nMENU\n");
      printf("\n1.Searching\n2.Count the number of nodes\n3.Traverse\n4.Exit");
      printf("\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
        case 1:
                  search(head);
                  break;
        case 2:
                  count(head);
                  break;
        case 3:
                  traverse(head);
                  break;
        case 4:
        exit(0);
        default:
        printf("Invalid choice. Please enter 1-4\n");
      }}}
struct node* create(struct node *head)
{
   int n, item, i;
   struct node *temp, *first;
   printf("Enter the number of nodes you want to create: ");
   scanf("%d", &n);
   if (n <= 0) {
      printf("Number of nodes should be greater than zero.\n");
      return head;
   }
```

```
    for (i = 1; i <= n; i++) {
        printf("Enter the item for node %d: ", i);
        scanf("%d", &item);
        temp = (struct node*)malloc(sizeof(struct node));
        if (temp == NULL) {
            printf("Memory insufficient\n");
            break;
        }
        temp->data = item;
        temp->next = NULL;
        temp->prev=first;
        first->next = temp;
        first = temp;
    }
    return head;
}
void search(struct node *head)
{
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    int item, flag = 0, pos = 1;
    printf("Enter the element you want to search: ");
    scanf("%d", &item);
    struct node *ptr = head;
    while (ptr != NULL) {
        if (ptr->data == item) {
            flag = 1;
            printf("Item present at position %d\n", pos);
            break;
        }
        ptr = ptr->next;
        pos++;
    }
    if (flag == 0) {
        printf("Item not found\n");
    }}
void count(struct node *head)
{
    int count = 0;
    struct node *ptr = head;
    if (ptr == NULL) {
        printf("List is empty\n");
        return;
    }
    while (ptr != NULL){
        count++;
        ptr = ptr->next;
```

```
    }
    printf("\nCount of nodes is %d",count);
}
void traverse(struct node *head)
{
    struct node *ptr = head;
    if (ptr == NULL) {
        printf("List is empty\n");
        return;
    }
    while (ptr != NULL){
        printf("%d <-> ", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}
```

Output

```
Creating a linked list:
Enter the number of nodes you want to create: 3
Enter the item for node 1: 4
Enter the item for node 2: 5
Enter the item for node 3: 6

MENU
1.Searching
2.Count the number of nodes
3.Traverse
4.Exit
Enter your choice: 1
Enter the element you want to search: 4
Item present at position 1


MENU
1.Searching
2.Count the number of nodes
3.Traverse
4.Exit
Enter your choice: 1
Enter the element you want to search: 8
Item not found

MENU
1.Searching
2.Count the number of nodes
3.Traverse
4.Exit
```

```
Enter your choice: 2

Count of nodes is 3
MENU
1.Searching
2.Count the number of nodes
3.Traverse
4.Exit
Enter your choice: 3
4 <-> 5 <-> 6 <-> NULL

MENU
1.Searching
2.Count the number of nodes
3.Traverse
4.Exit
Enter your choice: 4
```

Experiment 17 **Date: 04/12/2024**

Doubly Linked List- Insertion & Deletion

Aim:

To implement the following operations on a doubly linked list

a) Creation
b) Insert a node at first position
c) Insert a node at last
d) Delete a node from first position
e) Delete a node from last
f) Traversal

Algorithm:

Declare the structure node

struct node

1 Declare data, struct node *next, *prev

main()

1 Start
2 Declare ch, struct node *head.
3 Set head=NULL
4 To create a list of nodes call create(head)
5 Display choices.
6 Read option ch.
    If ch==1 call insertbeg(head)
    If ch==2 call insertlast(head)
    If ch==3 call deletebeg(head)
    If ch==4 call deletelast(head)
    If ch==5 call traverse(head)
    If ch==6 exit from the program
7 Repeat steps 1 and 6 while(1)
8 Stop

struct node* create(struct node *head)

1 Start.
2 Declare item, n, i, struct node *temp, *first.
3 Read the number of nodes that wanted to create.
4 If n<=0
    Print “Number of nodes should be greater than zero”
    Go to step 6
5 For i=1 to i<=n
    Read the item for node
    Allocate memory for *temp
    If temp=NULL
        Print “Memory insufficient”
        Go to step 7
  Set temp->data=item

Set temp->next=NULL
Set temp->prev=first
Set first->next=temp
Set first=temp
6 Return head
7 Exit.

struct node* insertbeg(struct node *head)

1 Start.
2 Declare item.
3 Allocate memory for struct node *temp.
4 If temp=NULL
Print “Memory insufficient”
Go to step 9
5 Read the item that wanted to insert.
6 Set temp->data=item
7 If head=NULL
Set temp->next=NULL
Set temp->prev=NULL
Else
Set temp->prev=NULL
temp->next=head
head->prev=temp
8 Set head=temp
9 Return head
10 Exit.

struct node* insertlast(struct node *head)

1 Start.
2 Allocate memory for struct node *temp.
3 If temp=NULL
Print “Memory insufficient”
Go to step 8
4 Read the item that wanted to insert.
5 Set temp->data=item
6 Set temp->next=NULL
7 If head=NULL
Set temp->next=NULL
Set temp->prev=NULL
Else
Set *ptr=head
While ptr->next != NULL
Set ptr=ptr->next
Set ptr->next=temp
8 Return head
9 Exit.

struct node* deletebeg(struct node *head)

1 Start.

2 Allocate memory for struct node *ptr.
3 If head=NULL
  Print “Memory insufficient”
  Go to step 7
4 Set ptr=head
5 Set head=head->next
6 Free(ptr)
7 Return head
8 Exit.

struct node* deletelast(struct node *head)

1 Start.
2 If head=NULL
  Print “Memory insufficient”
  Go to step 4
3 If head->next=NULL
  Set head=NULL
  Free(head)
  Else
  Set *ptr=head
  While ptr->next != NULL
  Set *ptr1=ptr
  Set ptr=ptr->next
  Set ptr1->next=NULL
  Free(ptr)
4 Return head
5 Exit.

void traverse (struct node *head)

1 Start.
2 Declare struct node *ptr
3 Set ptr=head
4 If ptr=NULL
  Print “List is empty”
  Go to step 6
5 While ptr!=NULL
  Print ptr<->data
  Set ptr=ptr->next
  Print “NULL”
6 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *prev;
   int data;
```

```
    struct node *next;
};
struct node* create(struct node *head);
struct node* insertbeg(struct node *head);
struct node* insertlast(struct node *head);
struct node* deletebeg(struct node *head);
struct node* deletelast(struct node *head);
void traverse(struct node *head);
void main()
{
    struct node *head = NULL;
    int ch, item;
    printf("Creating a linked list:\n");
    head=create(head);
    while(1)
    {
        printf("\nMENU\n");
        printf("\n1.Insert a node at beginning\n2.Insert a node at end\n3.Delete a
node from beginning\n4.Delete a node from end\n5.Traverse\n6.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
          case 1:
                    head = insertbeg(head);
                    break;
          case 2:
                    head = insertlast(head);
                    break;
            case 3:
                    head = deletebeg(head);
                    break;
          case 4:
                    head = deletelast(head);
                    break;
          case 5:
                    traverse(head);
                    break;
          case 6:
          exit(0);
          default:
          printf("Invalid choice. Please enter 1-6\n");
        }
    }
}
struct node* create(struct node *head)
{
    int n, item, i;
    struct node *temp, *first;
```

```
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
       printf("Number of nodes should be greater than zero.\n");
       return head;
    }
    printf("Enter the item for node 1: ");
    scanf("%d", &item);
    head = (struct node*)malloc(sizeof(struct node));
    if (head == NULL)
    {
       printf("Memory insufficient\n");
       exit(1);
    }
    head->prev=NULL;
    head->data = item;
    head->next = NULL;
    first = head;
    for (i = 2; i <= n; i++)
    {
       printf("Enter the item for node %d: ", i);
       scanf("%d", &item);
       temp = (struct node*)malloc(sizeof(struct node));
       if (temp == NULL) {
          printf("Memory insufficient\n");
          break;
       }
       temp->data = item;
       temp->next = NULL;
       temp->prev=first;
       first->next = temp;
       first = temp;
    }
    return head;
}
struct node* insertbeg(struct node *head)
{
    int item;
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    if (temp == NULL)
    {
       printf("Memory insufficient\n");
       return head;
    }
    printf("Enter the item you want to insert: ");
    scanf("%d", &item);
    temp->data = item;
    if(head == NULL)
    {
```

```
        temp->next = NULL;
        temp->prev = NULL;
      }
      else
      {
        temp->prev = NULL;
        temp->next = head;
        head->prev = temp;
      }
      head = temp;
      printf("One node is inserted at the beginning\n");
      return head;
    }
    struct node* insertlast(struct node *head)
    {
      int item;
      struct node *temp = (struct node *)malloc(sizeof(struct node));
      if (temp == NULL)
      {
        printf("Memory insufficient\n");
        return head;
      }
      printf("Enter the item you want to insert: ");
      scanf("%d", &item);
      temp->data = item;
      temp->next = NULL;
      if (head == NULL)
      {
        temp->next = NULL;
        temp->prev = NULL;
      }
      else
      {
        struct node *ptr = head;
        while (ptr->next != NULL)
        {
          ptr = ptr->next;
        }
        ptr->next = temp;
      }
      printf("One node is inserted at the end\n");
      return head;
    }
    struct node* deletebeg(struct node *head)
    {
      struct node *ptr = (struct node *)malloc(sizeof(struct node));
      if (head == NULL)
      {
        printf("Memory insufficient\n");
```

```
        return head;
    }
    ptr = head;
    head = head->next;
    free(ptr);
    printf("One node is deleted from the beginning\n");
    return head;
}
struct node* deletelast(struct node *head)
{
    if (head == NULL)
    {
        printf("Memory insufficient\n");
        return head;
    }
    if (head->next == NULL)
    {
        head = NULL;
        free(head);
    }
    else
    {
        struct node *ptr = head;
        struct node *ptr1;
        while (ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr->next;
        }
        ptr1->next = NULL;
        free(ptr);

    }
    printf("One node is deleted from the end\n");
    return head;
}
void traverse(struct node *head)
{
    struct node *ptr = head;
    if (ptr == NULL)
    {
        printf("List is empty\n");
        return;
    }
    while (ptr != NULL)
     {
        printf("%d <-> ", ptr->data);
        ptr = ptr->next;
    }
```

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

Output

```
Creating a linked list:
Enter the number of nodes you want to create: 5
Enter the item for node 1: 2
Enter the item for node 2: 3
Enter the item for node 3: 4
Enter the item for node 4: 5
Enter the item for node 5: 1

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 5
2 <-> 3 <-> 4 <-> 5 <-> 1 <-> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 3
One node is deleted from the beginning

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 4
One node is deleted from the end

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
```

```
6.Exit
Enter your choice: 5
3 <-> 4 <-> 5 <-> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 1
Enter the item you want to insert: 9
One node is inserted at the beginning

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 2
Enter the item you want to insert: 0
One node is inserted at the end


MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 5
9 <-> 3 <-> 4 <-> 5 <-> 0 <-> NULL

MENU
1.Insert a node at beginning
2.Insert a node at end
3.Delete a node from beginning
4.Delete a node from end
5.Traverse
6.Exit
Enter your choice: 6
```