Name. Gagandeep Singh Sohal

Course. Data structure and Algorithms Program. B.tech CSE

Sap id. 500119458

Batch no. 32

Roll no. R2142231082

# EXPERIMENT -1

## Quest. Find sum of all array elements using recursion. Ans.

# include <stdio.h>

int sum\_array\_elements(int \*, int); int main()

{

int arr[] = {1, 2, 3, -56, -0, -9, -98, 376, -56, -104, -98, -11};

int n = sizeof(arr)/sizeof(arr[0]);

int sum = sum\_array\_elements(arr, n); printf("The sum of array elements : %d\n", sum); return 0;

}

int sum\_array\_elements(int arr[12], int n)

{

if (n<=0){ return 0;

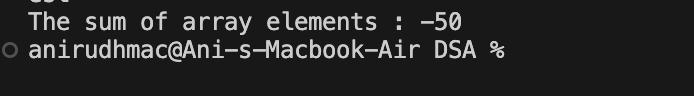
}

else{

return arr[n-1] + sum\_array\_elements(arr, n-1);

}

}



**Quest. Create an array 'al' with 'n' elements. Insert an element in i" position of 'al' and also delete an element from j" position of ‘al’.**

### Ans.

#include <stdio.h> #define MAX\_SIZE 100

void insertElement(int \*, int \*, int, int); void deleteElement(int \*, int \*, int);

int main() {

int arr[MAX\_SIZE]; int n;

int i, j;

int element;

printf("Enter the number of elements in array: "); scanf("%d", &n);

printf("Enter the elements of the array:\n"); for (int k = 0; k < n; k++) {

scanf("%d", &arr[k]);

}

printf("Enter the position to insert the element: "); scanf("%d", &i);

printf("Enter the element to insert: "); scanf("%d", &element);

insertElement(arr, &n, i, element);

printf("Array after insertion:\n"); for (int k = 0; k < n; k++) {

printf("%d ", arr[k]);

}

printf("\n");

printf("Enter the position to delete the element: "); scanf("%d", &j);

deleteElement(arr, &n, j);

printf("Array after deletion:\n"); for (int k = 0; k < n; k++) {

printf("%d ", arr[k]);

}

printf("\n");

return 0;

}

void insertElement(int a[], int \*n, int i, int element) { if (\*n >= MAX\_SIZE) {

printf("Array is full. Cannot insert element.\n");

return;

}

if (i < 0 || i > \*n) {

printf("Invalid position for insertion.\n"); return;

}

// Shift elements to make space for new element for (int j = \*n - 1; j >= i; j--) {

a[j + 1] = a[j];

}

a[i] = element;

// increasing size of array by 1 (\*n)++;

}

void deleteElement(int a[], int \*n, int j) { if (\*n <= 0) {

printf("Array is empty. Cannot delete element.\n"); return;

}

if (j < 0 || j >= \*n) {

printf("Invalid position for deletion.\n"); return;

}

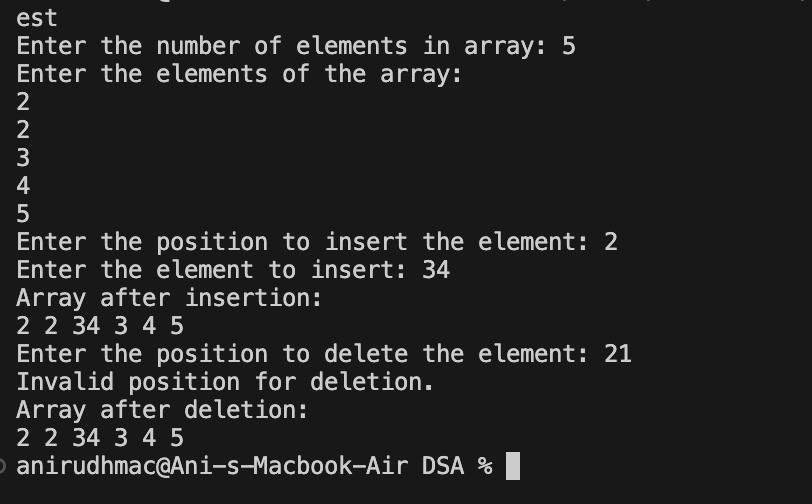
for (int i = j; i < \*n - 1; i++) {

// left shifting a[i] = a[i + 1];

}

// decreasing size of array (\*n)--;

}



## Quest. Convert uppercase string to lowercase using for loop.

**Ans.** #include<stdio.h> int main()

{

char str[] = "Upper Case Letter Is Formed"; printf("Uppercase string is: %s\n", str);

for(int i = 0; str[i] != '\0'; i++){

if (str[i] >= 'A' && str[i]<='Z')

{

str[i] = str[i] + 32;

}

}

printf("Lowercase string be: %s\n", str); return 0;

}

## Quest.Find the sum of rows and columns of matrix of given order (row x column).

### Ans.

#include <stdio.h>

# define MAX\_SIZE 100

int SumRowElements(int arr[][MAX\_SIZE], int, int, int);

int SumColumnElements(int arr[][MAX\_SIZE], int, int, int);

int main() {

int row, column;

int arr[MAX\_SIZE][MAX\_SIZE]; // 2D array

int sum\_row, sum\_column, rowIndex, columnIndex;;

printf("Enter the number of rows and columns (space-separated): "); scanf("%d %d", &row, &column);

printf("Enter the elements of the 2D array:\n"); for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) { scanf("%d", &arr[i][j]);

}

}

printf("Enter the row index for sum: "); scanf("%d", &rowIndex);

printf("Enter the column index for sum: "); scanf("%d", &columnIndex);

sum\_row = SumRowElements(arr, row, column, rowIndex); printf("Sum of elements in row %d: %d\n", rowIndex, sum\_row);

sum\_column = SumColumnElements(arr, row, column, columnIndex);

printf("Sum of elements in column %d: %d\n", columnIndex, sum\_column);

return 0;

}

int SumRowElements(int arr[][MAX\_SIZE], int row, int column, int rowIndex) { int sum = 0;

for (int j = 0; j < column; j++) { sum += arr[rowIndex][j];

}

return sum;

}

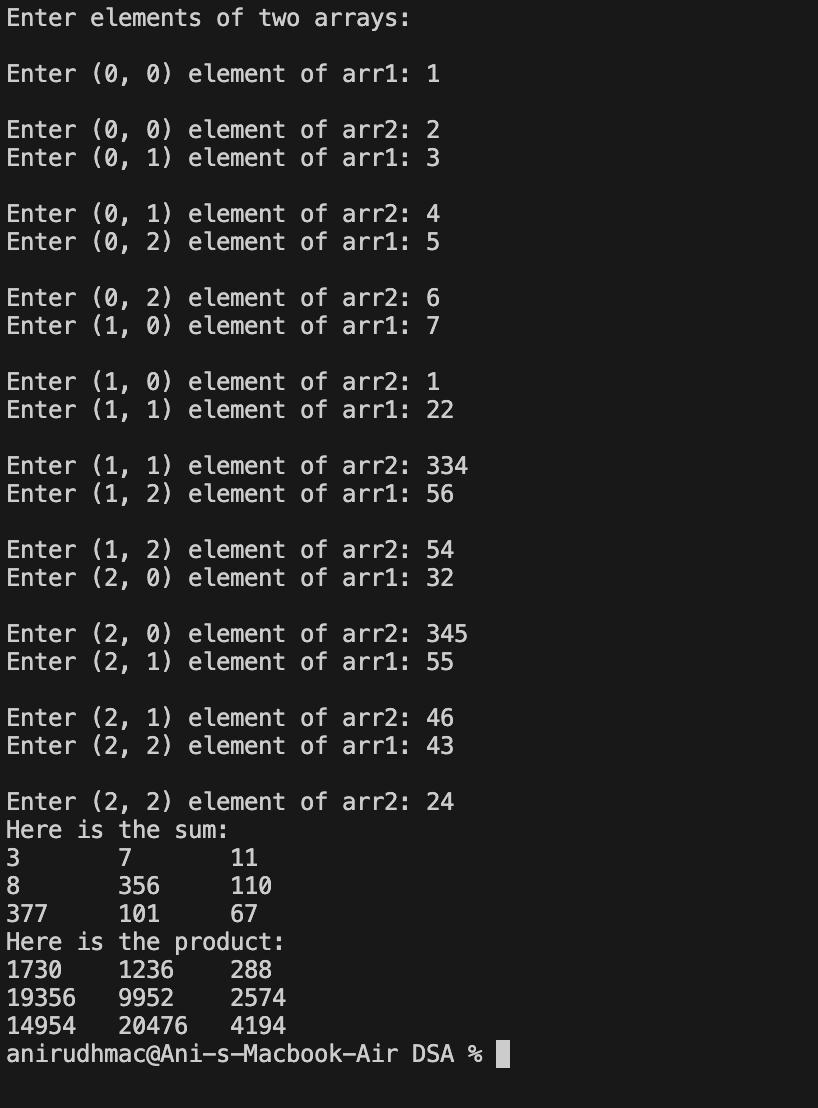
int SumColumnElements(int arr[][MAX\_SIZE], int row, int column, int columnIndex) { int sum = 0;

for (int i = 0; i < row; i++) { sum += arr[i][columnIndex];

}

return sum;

}



## Quest. Find the product of two matrices using pointers.

### Ans.

# include <stdio.h> int main()

{

int arr1[3][3];

int arr2[3][3];

int sum[3][3];

int mul[3][3];

printf("Enter elements of two arrays: \n\n"); for (int i =0;i<3;i++){

for(int j=0;j<3;j++){

printf("Enter (%d, %d) element of arr1: ", i, j); scanf("%d", &arr1[i][j]);

printf("\nEnter (%d, %d) element of arr2: ", i, j); scanf("%d", &arr2[i][j]);

}

}

for (int i=0; i<3; i++){ for(int j=0; j<3; j++){

sum[i][j] = arr1[i][j] + arr2[i][j];

}

}

printf("Here is the sum: \n"); for (int i =0;i<3;i++){

for(int j=0;j<3;j++){ printf("%d\t", sum[i][j]);

}

}

for (int i =0;i<3;i++){ for(int j=0;j<3;j++){

int sum=0;

for(int k =0;k<3;k++){ // here k is less than column number. sum = sum + arr1[i][k]\*arr2[k][j];

mul[i][j] = sum;

}

}

}

printf("Here is the product: \n"); for (int i =0;i<3;i++){

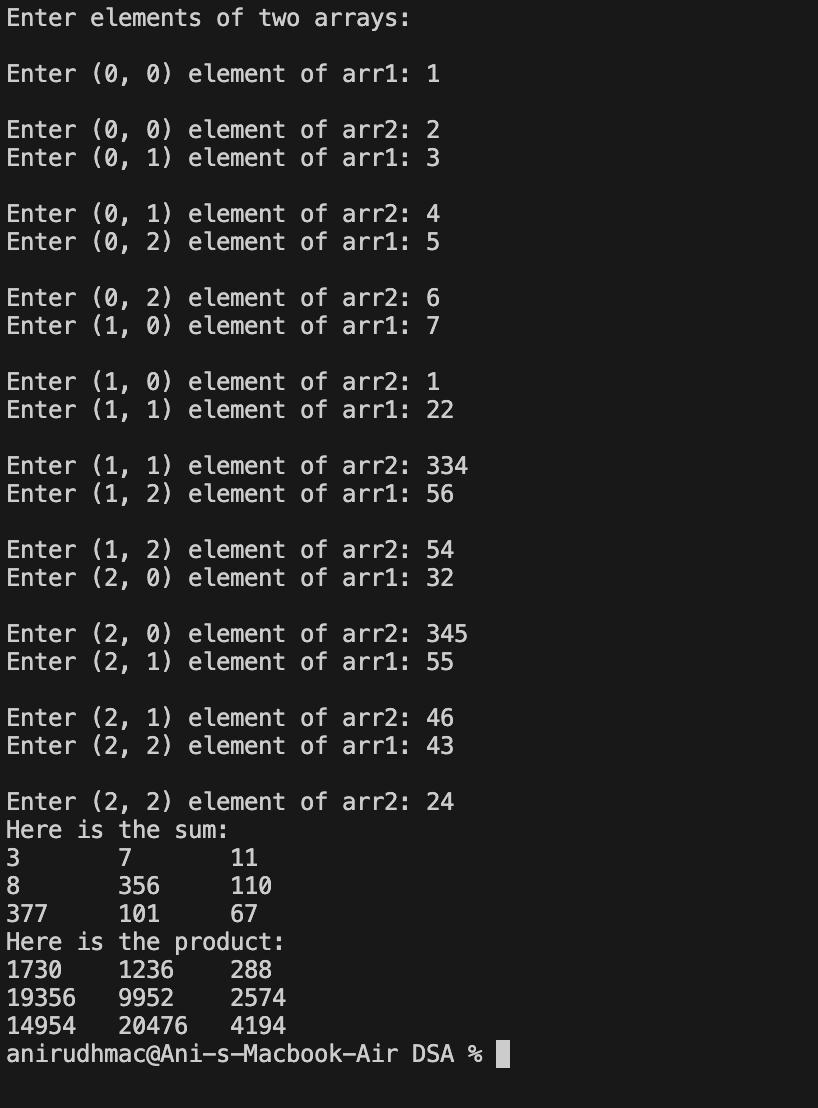
for(int j=0;j<3;j++){ printf("%d\t", mul[i][j]);

}

printf("\n");

}

}



## Quest. Store 'n' numbers (integers or real) in an array. Conduct a linear search for a given number and report success or failure in the form of a suitable message.

### Ans.

#include <stdio.h>

int linear\_search(int \*, int, int); int main() {

int n, i, key, result;

printf("Enter the number of elements in the array: "); scanf("%d", &n);

int arr[n+1];

printf("Enter %d elements:\n", n); for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Enter the number to search: "); scanf("%d", &key);

result = linear\_search(arr, n, key); if (result != -1) {

printf("Element %d found at index %d.\n", key, result);

} else

{

printf("Element %d not found in the array.\n", key);

}

return 0;

}

int linear\_search(int arr[], int n, int key) { for (int i = 0; i < n; i++) {

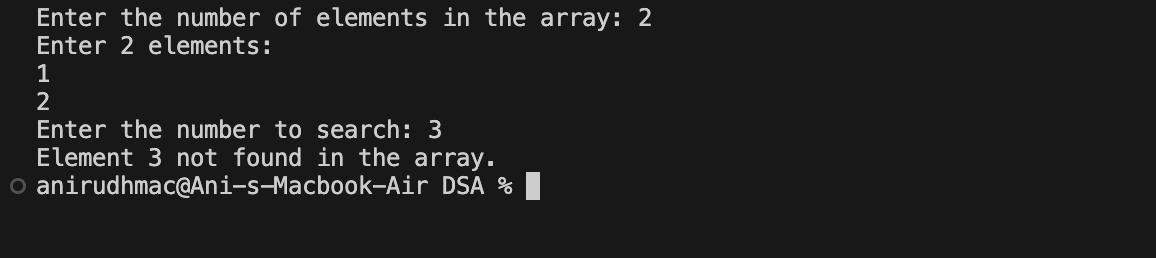
if (arr[i] == key) { return i;

}

}

return -1;

}



# EXPERIMENT -2

## Quest. Implement single Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list.

**Ans.** #include<stdio.h> #include<stdlib.h> struct node

{

int data;

struct node \*link;

};

void createlist(int size,struct node \*head); void display(struct node \*head);

void addatbeginning(struct node\*head); void addatend(int n,struct node \*head); void addafter(struct node \*head);

void addbefore(struct node \*head);

void deletebeginning(struct node \*head); void deleteend(struct node \*head);

void deletegivennode(struct node \*head); void deleteafter(struct node \*head);

void deleteentirelist(struct node \*head); void reverselist(struct node \*head); void middleelement(struct node \*head); void countinlist(struct node \*head);

int main()

{

int n;

printf("Enter the number of nodes"); scanf("%d",&n);

struct node \*head;

head=(struct node\*)malloc(sizeof(struct node)); printf("Enter the data in head\n "); scanf("%d",&head->data);

head->link=NULL; createlist(n,head);

printf("Enter your choice"); int choice; scanf(“%d",&choice);

switch (choice)

{

case(1):

printf("You have already created a list\nTHANK YOU !!"); break;

case(2):

display(head); break;

case(3):

addatbeginning(head); break;

case(4):

addatend(n,head); break;

case(5):

addbefore(head); break;

case(6):

addafter(head); break;

case(7):

deletebeginning(head); break;

case(8):

deleteend(head); break;

case(9):

deletegivennode(head); break;

case(10): deleteafter(head); break;

case(11): deleteentirelist(head); break;

case(12): reverselist(head); break;

case(13): middleelement(head); break;

case(14): countinlist(head); break;

default:

printf("You Entered wrong choice"); break;

}

return 0;

}

void createlist(int size,struct node\*head)

{

struct node \*ptr,\*newnode; ptr=head;

for(int i=2;i<=size;i++)

{

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the value of node %d\n",i); scanf("%d",&newnode->data);

newnode->link=NULL; ptr->link=newnode; ptr=newnode;

}

}

void display(struct node\*head)

{

struct node \*ptr; ptr=head; while(ptr!=NULL)

{

printf("%d\n",ptr->data); ptr=ptr->link;

}

}

void addatbeginning(struct node\*head)

{

struct node \*ptr,\*newnode; if (newnode==NULL)

printf("Unable to allocate memory");

else ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the value of new node"); scanf("%d",&newnode->data);

newnode->link=ptr; head=newnode; display(head);

}

void addatend(int size,struct node \*head)

{

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node));

printf("Enter the value of newnode"); scanf("%d", &newnode->data); newnode->link=NULL;

while(ptr->link!= NULL)

{

ptr=ptr->link;

}

ptr->link=newnode; display(head);

}

void addafter(struct node \*head)

{

int position;

printf("Enter a position after which you want to add node\n"); scanf("%d",&position);

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); newnode->link=NULL;

printf("Enter the data in node"); scanf("%d",&newnode->data); int i=0;

while(i!=position)

{

ptr=ptr->link; i++;

}

newnode->link=ptr->link; ptr->link=newnode; display(head);

}

void addbefore(struct node \*head)

{

int position;

printf("Enter the position before which you want to delete a node"); scanf("%d",&position);

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the value at node"); scanf("%d",&newnode->data);

newnode->link=NULL; int i =0; position=position-1; while(i<position)

{

ptr=ptr->link; i++;

}

newnode->link=ptr->link; ptr->link=newnode; display(head);

}

void deletebeginning(struct node \*head)

{

struct node \*ptr; ptr=head; head=ptr->link; ptr->link=NULL; free(ptr); display(head);

}

void deleteend(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

while(ptr->link!=NULL)

{

temp=ptr; ptr=ptr->link;

}

temp->link=NULL;

free(ptr); display(head);

}

void deletegivennode(struct node \*head)

{

int position;

printf("Enter the node no you want to delete "); scanf("%d",&position);

struct node \*ptr,\*temp; ptr=head;

int i=0; while(i!=position)

{

temp=ptr; ptr=ptr->link; i++;

}

temp->link=ptr->link; ptr->link=NULL; free(ptr); display(head);

}

void deleteafter(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

int position;

printf("Enter a position after which you want to delete a node\n"); scanf("%d",&position);

int i=0; position=position+1; while(i<position)

{

temp=ptr; ptr=ptr->link; i++;

}

temp->link=ptr->link; ptr->link=NULL;

free(ptr); display(head);

}

void deleteentirelist(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

while(ptr->link!=NULL)

{

temp=ptr; ptr=ptr->link;

temp->link=NULL; free(temp);

}

display(head);

}

void reverselist(struct node \*head)

{

struct node \*ptr; ptr=head;

if(ptr)

{

reverselist(ptr->link); printf("%d ",ptr->data);

}

}

void middleelement(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

temp=head; int count=0;

while(ptr->link!=NULL)

{

ptr=ptr->link; count++;

}

if(count%2==0)

{

count=(count+1)%2;

}

else

{

count=count%2;

}

for(int i=0;i<=count;i++)

{

temp=temp->link;

}

printf("The value of middle element is %d",temp->data);

}

void countinlist(struct node \*head)

{

struct node \*ptr; ptr=head;

int count=0;

while(ptr->link!=NULL)

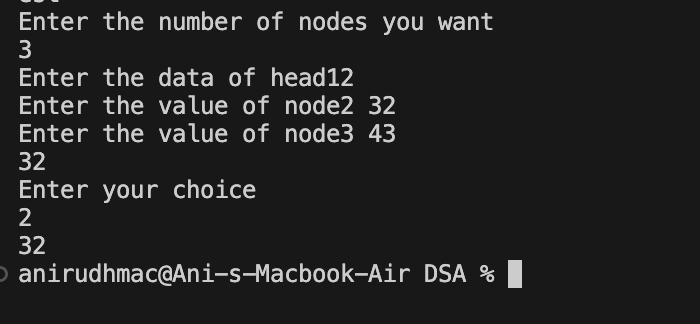
{

ptr=ptr->link; count++;

}

printf("The count of list is %d",count);

}



## Quest. Using single linked list and functions implement Stack and its operations like insert, delete, and display.

### Ans.

# include <stdio.h> # include <stdlib.h> struct node{

int data;

struct node \*link;

};

struct node \*top=0; // our head has become top as a pointer

void Push\_ll(int); void display\_ll(); void peek\_ll(); void pop\_ll();

int main()

{

int value, choice, option, answer

do {

printf("Enter your choice:\n\n\t1.Pushing element in stack.\n\t2.Display the stack.

\n\t. 3.Peeking on the stack.\n\t4.Pop the element.\n"); scanf("%d", &option);

switch(option){ case 1:

do {

printf("Enter the value: \n"); scanf("%d", &value); Push\_ll(value);

printf("Want to add more ? (0/1): "); scanf("%d", &choice);

}while(choice == 1); break;

case 2:

display\_ll(); break;

case 3:

peek\_ll(); break;

case 4:

pop\_ll();

**}**

printf("\nDo you want to continue with choice ? (1/0)\n"); scanf("%d", &answer);

}while(answer == 1);

return 0;

}

void Push\_ll(int x)

{

struct node \*newnode = (struct node \*)malloc(sizeof(struct node)); newnode->data = x;

newnode->link = top; top = newnode;

}

void display\_ll()

{

struct node \*temp; temp = top;

if (top == NULL)

{

printf("\nList is Empty!");

}

else

{

while(temp != 0){ printf("\n\t%d\n", temp->data); temp = temp->link;

}

}

}

void peek\_ll()

{

if (top==0){

printf("\nList is Empty!");

}

else{

printf("\nTop element is %d\n", top->data);

}

}

void pop\_ll()

{

struct node \*temp; temp = top;

if (top == 0){ printf("Underflow!");

}

else

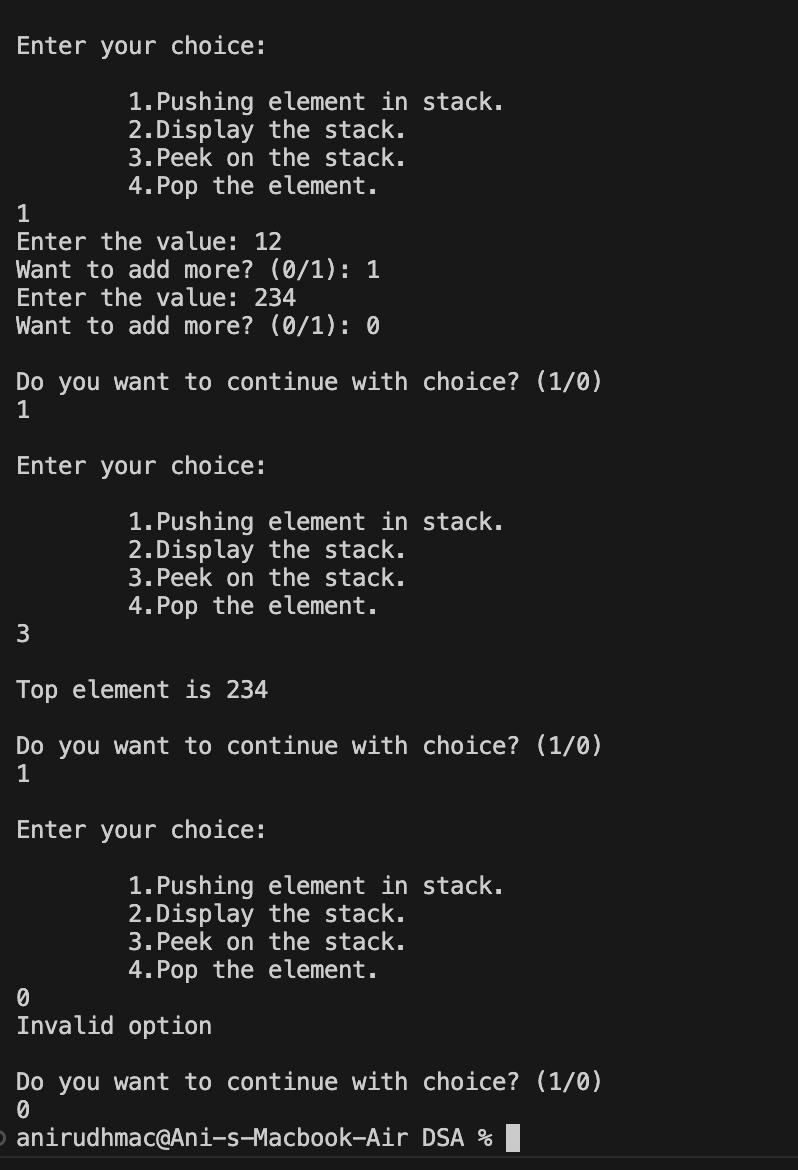
{

printf("\n%d is poped out!\n", top->data); top=top->link;

free(temp);

}

}



## Quest. Implement Circular Linked List and its operations.

**Ans.** #include<stdio.h> #include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

void creation(int n,struct node \*head); void display(struct node \*head);

void addnodebeginning(struct node \*head);

int main()

{

int n;

printf("Enter the number of nodes you want\n"); scanf("%d",&n);

struct node \*head;

head=(struct node\*)malloc(sizeof(struct node)); printf("Enter the data of head"); scanf("%d",&head->data);

head->next=NULL; creation(n,head);

printf("Enter your choice\n"); int choice; scanf("%d",&choice); switch(choice)

{

case(1):

printf("YOU HAVE CREATED ALREADY");

break; case(2): display(head); break; case(3):

addnodebeginning(head); break;

case(4): break;

case(5): break; case(6): break; case(7): break; case(8): break;

}

}

void creation(int n,struct node \*head)

{

struct node \*newnode,\*ptr; ptr=head;

for(int i=2;i<=n;i++)

{

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the value of node%d ",i); scanf("%d",&newnode->data);

newnode->next=NULL; ptr->next=newnode; ptr=newnode;

}

ptr->next=head; display(head);

}

void display(struct node \*head)

{

struct node \*ptr; ptr=head; ptr=ptr->next;

while(ptr->next!=head)

{

printf("%d\n",ptr->data); ptr=ptr->next;

}

}

void addnodebeginning(struct node \*head)

{

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the data of newnode"); scanf("%d",&newnode->data);

while(ptr->next!=head)

{

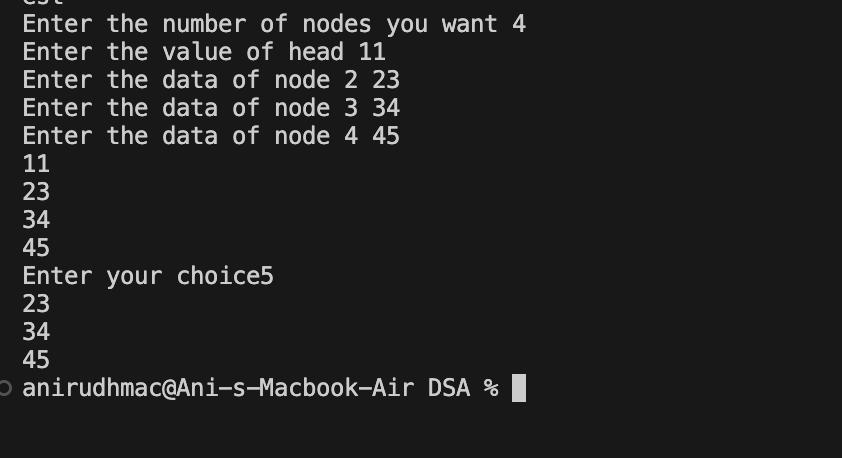
ptr=ptr->next;

}

newnode->next=ptr; head=newnode;

display(head);

}



## Quest. Implement Doubly Linked List and its operations.

**Ans.** #include<stdio.h> #include<stdlib.h> struct node

{

struct node \*previous; int data;

struct node \*next;

};

void creation(int n,struct node \*head); void display(struct node \*head);

void insertionbeginning(struct node \*head); void insertionend(struct node \*head);

void insertionbefore(struct node \*head); void insertionafter(struct node \*head); void deletionbeginning(struct node \*head); void deletionend(struct node \*head);

void reverse(struct node \*head);

int main()

{

int n;

printf("Enter the number of nodes you want "); scanf("%d",&n);

struct node \*head;

head=(struct node\*)malloc(sizeof(struct node)); printf("Enter the value of head "); scanf("%d",&head->data);

head->previous=NULL; head->next=NULL;

creation(n,head); display(head);

int choice;

printf("Enter your choice"); scanf("%d",&choice);

switch(choice)

{

case(1): insertionbeginning(head); break;

case(2): insertionend(head); break;

case(3): insertionbefore(head); break;

case(4): insertionafter(head); break;

case(5): deletionbeginning(head); break;

case(6): deletionend(head); break;

case(7): reverse(head); break;

}

}

void creation(int n,struct node \*head)

{

struct node \*ptr,\*newnode; ptr=head;

for(int i=2;i<=n;i++)

{

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter the data of node %d ",i); scanf("%d",&newnode->data);

newnode->next=NULL; ptr->next=newnode; newnode->previous=ptr; ptr->previous=NULL; ptr=newnode;

}

}

void display(struct node \*head)

{

struct node \*ptr; ptr=head; while(ptr!=NULL)

{

printf("%d\n",ptr->data); ptr=ptr->next;

}

}

void insertionbeginning(struct node \*head)

{

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter data of new node"); scanf("%d",&newnode->data);

newnode->next=ptr->next; ptr->next=newnode;

ptr->previous=newnode; newnode->previous=NULL; head=newnode;

display(head);

}

void insertionend(struct node \*head)

{

struct node \*ptr,\*newnode; ptr=head;

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter data of new node"); scanf("%d",&newnode->data);

newnode->next=NULL; while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=newnode; newnode->previous=ptr;

display(head);

}

void insertionbefore(struct node \*head)

{

struct node \*ptr,\*temp,\*newnode; ptr=head;

temp=head; int position;

printf("Enter the position before you want to insert"); scanf("%d",&position);

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter data of new node"); scanf("%d",&newnode->data);

newnode->next=NULL; newnode->previous=NULL; for(int i=0;i<position;i++)

{

temp=ptr; ptr=ptr->next;

}

newnode->next=temp->next; temp->next=newnode;

ptr->previous=newnode; newnode->previous=temp;

display(head);

}

void insertionafter(struct node \*head)

{

struct node \*ptr,\*temp,\*newnode;

ptr=head; temp=head; int position;

printf("Enter the position after you want to insert"); scanf("%d",&position);

newnode=(struct node\*)malloc(sizeof(struct node)); printf("Enter data of new node"); scanf("%d",&newnode->data);

newnode->next=NULL; newnode->previous=NULL; for(int i=0;i<position;i++)

{

ptr=ptr->next;

}

temp=ptr->next;

newnode->next=ptr->next; ptr->next=newnode;

temp->previous=newnode; newnode->previous=ptr;

display(head);

}

void deletionbeginning(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

temp=head; temp=ptr->next; ptr->next=NULL;

temp->previous=NULL; head=temp;

free(ptr);

display(head);

}

void deletionend(struct node \*head)

{

struct node \*ptr,\*temp; ptr=head;

temp=head;

while(ptr->next!=NULL)

{

temp=ptr; ptr=ptr->next;

}

temp->next=NULL; ptr->previous=NULL; free(ptr);

display(head);

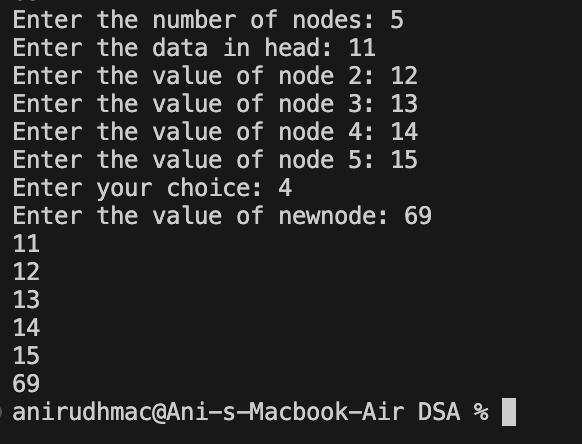
}

void reverse(struct node \*head)

{

struct node \*ptr; ptr=head;

}



# EXPERIMENT - 3

## Quest. Using array and functions implement Stack and its operations like insert, delete, and display.

**Ans.** #include<stdio.h> #include<stdlib.h> #define size 4

int arr[size]; int top=-1;

void push(); void display(); void pop(); void peek();

int main()

{

int choice;

printf("Ebter your choice "); scanf("%d",&choice);

switch (choice)

{

case(1):

push(); break; case(2):

pop(); break; case(3):

peek(); break; case(4):

break; default:

printf("You entered wrong choice"); break;

}

}

void push()

{

int value;

for(int i=0;i<size;i++)

{

printf("Enter the value of %d ",i); scanf("%d",&value);

}

if(top==size-1)

{

printf("Stack overflow");

}

else

{

top=top+1; arr[top]=value;

}

display();

}

void display()

{

for(int i=0;i<=top;i++)

{

printf("Elements of stack are %d=%d\n",i,arr[i]);

}

}

void pop()

{

int popped; if(top==-1)

{

printf("Underdlow");

}

else

{

popped=arr[top]; top--;

}

printf("Popped item is %d",popped);

}

void peek() //check the top element of stack

{

if(top==-1)

{

printf("Stack is empty");

}

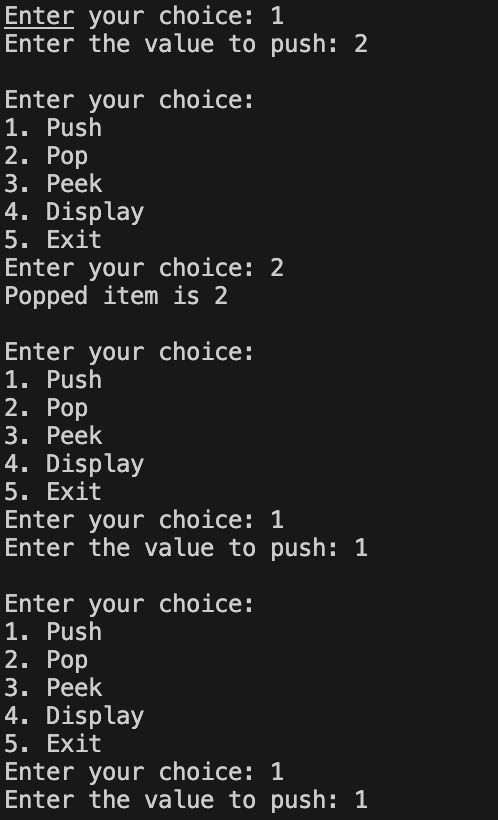
else

{

printf("Top most elemnt if %d",arr[top]);

}

}



## Quest. Reverse a string using stack.

### Ans.

# include <stdio.h> # include <stdlib.h> # include <limits.h> # include <string.h>

struct Stack { int top;

unsigned capacity; char \*array;

};

# define MAX\_SIZE 100

struct Stack \*CreateStack(unsigned); int IsFull(struct Stack \*);

int IsEmpty(struct Stack \*); void Push(struct Stack \*, char); char Pop (struct Stack \*);

void reverse(char \*);

// creating Stack

struct Stack \* CreateStack(unsigned capacity)

{

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack)); stack->capacity = capacity;

stack->top = -1;

// array of that size storing characters

stack->array = (char \*)malloc(stack->capacity \* sizeof(char));

return stack;

}

int IsFull(struct Stack \*stack)

{

return stack->top == stack->capacity - 1;

}

int IsEmpty(struct Stack \*stack)

{

return stack->top == -1;

}

void Push(struct Stack \*stack, char item)

{

// is full condition if (IsFull(stack)){

return;

}

stack->array[++stack->top] = item;

}

char Pop(struct Stack \*stack)

{

// is empty condition if (IsEmpty(stack)){

return '\0';

}

// returning elements one by one return stack->array[stack->top--];

}

void reverse(char str[])

{

int n = strlen(str);

struct Stack \*stack = CreateStack(n);

for(int i=0; i<n; i++){ Push(stack, str[i]);

}

for (int i = 0; i<n; i++){

str[i] = Pop(stack);

}

}

int main()

{

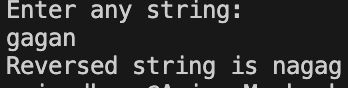
char str[MAX\_SIZE]; printf("Enter any string: \n"); scanf(" %[^\n]", str);

// char str[] = "Anirudh";

reverse(str);

printf("Reversed string is %s", str); return 0;

}



## Quest 3. Convert infix to postfix expression using stack and array.

### Ans.

#include <stdio.h> #include <stdlib.h> #include <ctype.h>

#define MAX\_SIZE 100 struct Stack {

int top;

char array[MAX\_SIZE];

};

void push(struct Stack \*stack, char ch); char pop(struct Stack \*stack);

int precedence(char ch); int isOperator(char ch);

void infixToPostfix(char infix[], char postfix[]);

int main()

{

char infix[MAX\_SIZE]; char postfix[MAX\_SIZE];

printf("Enter an infix expression: ");

// standard input devices

fgets(infix, MAX\_SIZE, stdin); infixToPostfix(infix, postfix); printf("Postfix expression: %s\n", postfix);

return 0;

}

void push(struct Stack \*stack, char ch)

{

if (stack->top == MAX\_SIZE - 1) { printf("Stack Overflow\n"); exit(1);

}

stack->array[++stack->top] = ch;

}

char pop(struct Stack \*stack)

{

if (stack->top == -1) { printf("Stack Underflow\n"); exit(1);

}

return stack->array[stack->top--];

}

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

int precedence(char ch)

{

switch (ch) { case '+':

case '-':

return 1; case '\*':

case '/':

return 2; default:

return -1;

}

}

void infixToPostfix(char infix[], char postfix[])

{

struct Stack stack; stack.top = -1;

int i = 0, j = 0;

while (infix[i] != '\0') {

// here ctype library is used if (isdigit(infix[i])) {

postfix[j++] = infix[i++];

}

else if (isOperator(infix[i])) {

while (stack.top != -1 && precedence(stack.array[stack.top]) >= precedence(infix[i])) {

postfix[j++] = pop(&stack);

}

push(&stack, infix[i++]);

}

else if (infix[i] == '(') { push(&stack, infix[i++]);

}

else if (infix[i] == ')') {

while (stack.top != -1 && stack.array[stack.top] != '(') { postfix[j++] = pop(&stack);

}

if (stack.top == -1) { printf("Invalid expression\n"); exit(1);

}

pop(&stack); // Discard the '(' i++;

}

else if (infix[i] == ' ' || infix[i] == '\t' || infix[i] == '\n') { i++; // Ignore whitespaces, tabs, and newlines

}

else {

printf("Invalid character in infix expression: %c\n", infix[i]); exit(1);

}

}

// Pop remaining operators from stack while (stack.top != -1) {

if (stack.array[stack.top] == '(') { printf("Invalid expression\n"); exit(1);

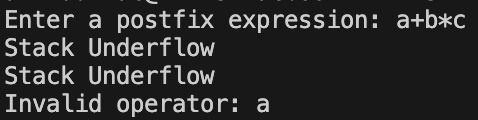
}

postfix[j++] = pop(&stack);

}

postfix[j] = '\0'; // Null-terminate the postfix string

}



## Quest. Evaluate postfix expression using stack and array. Ans.

#include <stdio.h> #include <stdlib.h> #include <ctype.h>

#define MAX\_SIZE 100 struct Stack {

int top;

int array[MAX\_SIZE];

};

void push(struct Stack \*stack, int num); int pop(struct Stack \*stack);

int evaluatePostfix(char postfix[]);

int main()

{

char postfix[MAX\_SIZE];

printf("Enter a postfix expression: "); fgets(postfix, MAX\_SIZE, stdin);

int result = evaluatePostfix(postfix); printf("Result: %d\n", result);

return 0;

}

void push(struct Stack \*stack, int num) { if (stack->top == MAX\_SIZE - 1) {

printf("Stack Overflow\n"); exit(1);

}

stack->array[++stack->top] = num;

}

int pop(struct Stack \*stack) { if (stack->top == -1) {

printf("Stack Underflow\n"); exit(1);

}

return stack->array[stack->top--];

}

int evaluatePostfix(char postfix[]) { struct Stack stack;

stack.top = -1;

int i = 0;

while (postfix[i] != '\0') { if (isdigit(postfix[i])) {

// Converting char to int push(&stack, postfix[i] - '0');

}

else if (postfix[i] == ' ' || postfix[i] == '\t' || postfix[i] == '\n') {

// Ignore whitespaces, tabs, and newlines

}

else {

int operand2 = pop(&stack); int operand1 = pop(&stack);

switch (postfix[i]) { case '+':

push(&stack, operand1 + operand2); break;

case '-':

push(&stack, operand1 - operand2); break;

case '\*':

push(&stack, operand1 \* operand2); break;

case '/':

if (operand2 == 0) { printf("Division by zero\n"); exit(1);

}

push(&stack, operand1 / operand2); break;

default:

printf("Invalid operator: %c\n", postfix[i]); exit(1);

}

}

i++;

}

if (stack.top == 0) {

// return the result

return stack.array[stack.top];

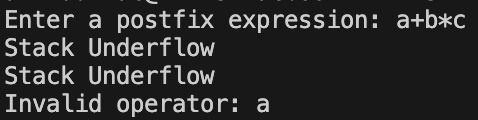
}

else {

printf("Invalid expression\n"); exit(1);

}

}



# EXPERIMENT - 4

## Quest. Using array and functions implement Queue data structure and its operations like insert, delete and display.

**Ans.** #include<stdio.h> #include<stdlib.h>

struct queue

{

int size; int f; int r; int \*arr;

};

void enqueue(struct queue \*q,int value); int dequeue(struct queue \*q);

int main()

{

struct queue q; q.size=10; q.f=q.r=-1;

q.arr=(int\*)malloc(q.size\*sizeof(int));

enqueue(&q,12);

printf("Dequeue element is %d",dequeue(&q));

}

void enqueue(struct queue \*q,int value)

{

q->r++;

q->arr[q->r]=value;

}

int dequeue(struct queue \*q)

{

int a = -1; q->f++;

return q->arr[q->f]**;**

**}**

## Quest. Check whether the string is palindrome or not using array and Queue.

### Ans.

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <ctype.h>

#define MAX\_SIZE 100 struct Queue {

int front, rear, size;

unsigned capacity; char \*array;

};

struct Queue\* createQueue(unsigned capacity); int isFull(struct Queue\* queue);

int isEmpty(struct Queue\* queue);

void enqueue(struct Queue\* queue, char item); char dequeue(struct Queue\* queue);

int isPalindrome(char str[]);

int main() {

char str[MAX\_SIZE];

printf("Enter a string: "); fgets(str, MAX\_SIZE, stdin);

if (isPalindrome(str)) {

printf("The string is a palindrome.\n");

} else {

printf("The string is not a palindrome.\n");

}

return 0;

}

struct Queue\* createQueue(unsigned capacity) {

struct Queue\* queue = (struct Queue\*) malloc(sizeof(struct Queue)); queue->capacity = capacity;

queue->front = queue->size = 0; queue->rear = capacity - 1;

queue->array = (char\*) malloc(queue->capacity \* sizeof(char)); return queue;

}

int isFull(struct Queue\* queue) {

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue) { return (queue->size == 0);

}

void enqueue(struct Queue\* queue, char item) { if (isFull(queue)) {

printf("Queue is full.\n"); return;

}

queue->rear = (queue->rear + 1) % queue->capacity; queue->array[queue->rear] = item;

queue->size = queue->size + 1;

}

char dequeue(struct Queue\* queue) { if (isEmpty(queue)) {

printf("Queue is empty.\n"); exit(1);

}

char item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity; queue->size = queue->size - 1;

return item;

}

int isPalindrome(char str[]) {

struct Queue\* queue = createQueue(MAX\_SIZE); int len = strlen(str);

for (int i = 0; i < len; i++) { if (isalpha(str[i])) {

enqueue(queue, tolower(str[i]));

}

}

for (int i = len - 1; i >= 0; i--) {

if (isalpha(str[i])) {

char ch = dequeue(queue); if (ch != tolower(str[i])) {

free(queue->array);

free(queue);

return 0; // not a palindrome

}

}

}

free(queue->array); free(queue);

return 1; // Palindrome

}



## Quest. Using array and functions implement Circular Queue data structure and its operations like insert, delete, and display.

### Ans.

# include <stdio.h> # include <stdlib.h>

# define N 5 int queue[N]; int front = -1; int rear = -1;

void Enqueue(int); void display(); void Dequeue(); void Peek();

void menu(); int main() {

int choice, value;

do {

menu();

scanf("%d", &choice);

switch(choice) { case 1:

printf("Enter the element: "); scanf("%d", &value); Enqueue(value);

break; case 2:

display();

break; case 3:

Dequeue(); break;

case 4:

Peek(); break;

case 0:

printf("Exiting...\n"); break;

default:

printf("Invalid choice! Please enter a valid option.\n");

}

} while(choice != 0);

return 0;

}

void Enqueue(int x)

{

if ((rear + 1) % N == front) { // Check if the queue is full printf("Overflow!\n");

return;

}

if (front == -1 && rear == -1) { // Empty queue front = rear = 0;

queue[rear] = x;

} else {

rear = (rear + 1) % N; queue[rear] = x;

}

}

void Dequeue(){

if (front == -1 && rear == -1){ printf("Underflow!\n"); return;

}

if (front == rear) { // Only one element front = rear = -1;

} else {

front = (front + 1) % N;

}

}

void display()

{

if (front == -1 && rear == -1){ printf("Queue is Empty!\n"); return;

}

int i = front; do {

printf("%d\n", queue[i]); i = (i + 1) % N;

} while (i != (rear + 1) % N);

}

void Peek(){

if (front == -1 && rear == -1){ printf("Queue is Empty!\n"); return;

}

printf("Front element: %d\n", queue[front]);

}

void menu() {

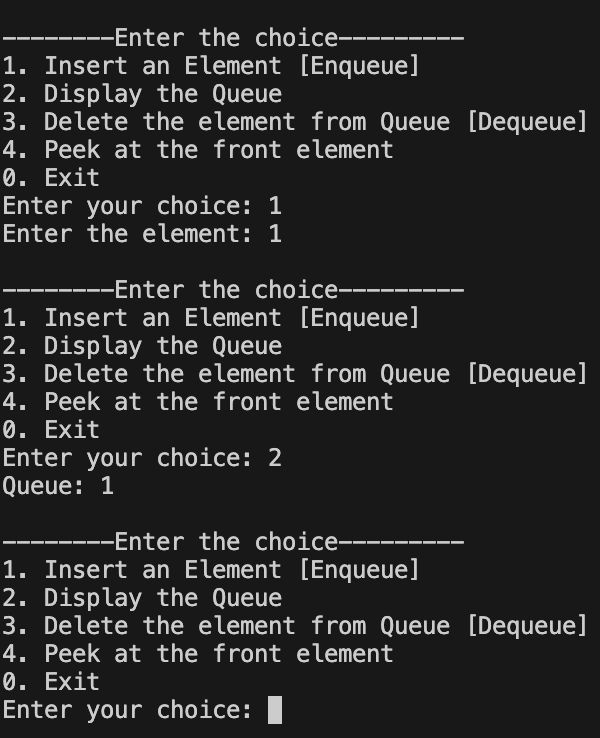
printf("\n--------Enter the choice \n");

printf("1. Insert an Element [Enqueue]\n"); printf("2. Display the Queue\n");

printf("3. Delete the element from Queue [Dequeue]\n"); printf("4. Peek at the front element\n");

printf("0. Exit\n"); printf("Enter your choice: ");

}



## Quest. Using array and functions implement a Stack using Queues.

### Ans.

#include <stdio.h> #include <stdlib.h>

#define MAX\_SIZE 100

struct Queue {

int front, rear, size; unsigned capacity; int \*array;

};

struct Queue\* createQueue(unsigned capacity); int isFull(struct Queue\* queue);

int isEmpty(struct Queue\* queue);

void enqueue(struct Queue\* queue, int item); int dequeue(struct Queue\* queue);

void push(struct Queue\* queue1, struct Queue\* queue2, int item); int pop(struct Queue\* queue1, struct Queue\* queue2);

void display(struct Queue\* queue);

int main() {

// so our capacity is MAX\_SIZE

struct Queue\* queue1 = createQueue(MAX\_SIZE); struct Queue\* queue2 = createQueue(MAX\_SIZE);

push(queue1, queue2, 1);

push(queue1, queue2, 2);

push(queue1, queue2, 3);

push(queue1, queue2, 4);

printf("Stack elements: "); display(queue1);

printf("Popped element: %d\n", pop(queue1, queue2));

printf("Stack elements after popping: "); display(queue1);

return 0;

}

struct Queue\* createQueue(unsigned capacity) {

struct Queue\* queue = (struct Queue\*) malloc(sizeof(struct Queue)); queue->capacity = capacity;

queue->front = queue->size = 0; queue->rear = capacity - 1;

queue->array = (int\*) malloc(queue->capacity \* sizeof(int)); return queue;

}

int isFull(struct Queue\* queue) {

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue) { return (queue->size == 0);

}

void enqueue(struct Queue\* queue, int item) { if (isFull(queue)) {

printf("Queue is full.\n"); return;

}

// circular one, preferred one

queue->rear = (queue->rear + 1) % queue->capacity; queue->array[queue->rear] = item;

queue->size = queue->size + 1;

}

int dequeue(struct Queue\* queue) { if (isEmpty(queue)) {

printf("Queue is empty.\n"); exit(1);

}

int item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity; queue->size = queue->size - 1;

return item;

}

void push(struct Queue\* queue1, struct Queue\* queue2, int item) { if (!isEmpty(queue1)) {

while (!isEmpty(queue1)) {

// pushing all elements from queue1 to queue2 enqueue(queue2, dequeue(queue1));

}

}

// enquing the new elment into queue1 enqueue(queue1, item);

while (!isEmpty(queue2)) {

// bringing back all elements to queue1 enqueue(queue1, dequeue(queue2));

}

}

// dequeue element from queue1

int pop(struct Queue\* queue1, struct Queue\* queue2) {

if (isEmpty(queue1)) { printf("Stack is empty.\n"); exit(1);

}

return dequeue(queue1);

}

void display(struct Queue\* queue) { int i = queue->front;

do {

printf("%d ", queue->array[i]); i = (i + 1) % queue->capacity;

} while (i != (queue->rear + 1) % queue->capacity); printf("\n");

}

# EXPERIMENT - 5

## Quest. Create a binary tree using an array/linked List.

**Ans.** #include<stdio.h> #include<stdlib.h>

struct node

{

struct node \*left; int data;

struct node \*right;

};

struct node \*creation(int value)

{

struct node \*n = (struct node\*)malloc(sizeof(struct node)); n->data = value;

n->left = NULL; n->right = NULL; return n;

}

void preOrder(struct node \*p); int main()

{

//creation of tree

struct node \*p = creation(2); struct node \*p1 = creation(1); struct node \*p2 = creation(3); struct node \*p3 = creation(4); struct node \*p4 = creation(25); struct node \*p5 = creation(6);

//linking in tree p->left = p1;

p->right = p2; p1->left = p3; p1->right = p4; p2->left = p5;

preOrder(p);

return 0;

}

void preOrder(struct node \*p)

{

if(p != NULL)

{

printf("%d ", p->data); preOrder(p->left); preOrder(p->right);

}

}



## Quest. Construct a Binary Tree and perform Inorder , Preorder and Postorder Traversal.

### Ans.

**PREORDER**

#include<stdio.h> #include<stdlib.h>

struct node

{

struct node \*left; int data;

struct node \*right;

};

struct node \*creation(int value)

{

struct node \*n = (struct node\*)malloc(sizeof(struct node)); n->data = value;

n->left = NULL; n->right = NULL; return n;

}

void preOrder(struct node \*p); int main()

{

//creation of tree

struct node \*p = creation(2); struct node \*p1 = creation(1); struct node \*p2 = creation(3); struct node \*p3 = creation(4); struct node \*p4 = creation(25); struct node \*p5 = creation(6);

//linking in tree p->left = p1;

p->right = p2; //2

p2->right = p3; // 1 3

p3->right = p4; // 4

p3->left = p5; // 6 25

preOrder(p);

return 0;

}

void preOrder(struct node \*p)

{

if(p != NULL)

{

printf("%d ", p->data); preOrder(p->left); preOrder(p->right);

}

}

### INORDER

#include<stdio.h> #include<stdlib.h>

struct node

{

struct node \*left; int data;

struct node \*right;

};

struct node \*creation(int value)

{

struct node \*n = (struct node\*)malloc(sizeof(struct node)); n->data = value;

n->left = NULL; n->right = NULL; return n;

}

void inOrder(struct node \*p); int main()

{

//creation of tree

struct node \*p = creation(2); struct node \*p1 = creation(1); struct node \*p2 = creation(3); struct node \*p3 = creation(4); struct node \*p4 = creation(25); struct node \*p5 = creation(6);

//linking in tree p->left = p1;

p->right = p2; p1->left = p3; p1->right = p4; p2->left = p5;

inOrder(p);

}

void inOrder(struct node \*p)

{

if(p != NULL)

{

inOrder(p->left); printf("%d ", p->data); inOrder(p->right);

}

}

### POSTORDER

#include<stdio.h> #include<stdlib.h>

struct node

{

struct node \*left; int data;

struct node \*right;

};

struct node \*creation(int value)

{

struct node \*n = (struct node\*)malloc(sizeof(struct node)); n->data = value;

n->left = NULL; n->right = NULL; return n;

}

void postOrder(struct node \*p); int main()

{

//creation of tree

struct node \*p = creation(2); struct node \*p1 = creation(1); struct node \*p2 = creation(3); struct node \*p3 = creation(4); struct node \*p4 = creation(25); struct node \*p5 = creation(6);

//linking in tree p->left = p1;

p->right = p2; p1->left = p3; p1->right = p4; p2->left = p5;

postOrder(p);

return 0;

}

void postOrder(struct node \*p)

{

if(p != NULL)

{

postOrder(p->left); postOrder(p->right); printf("%d ", p->data);

}

}

