*Index*

*Write a program in C language that:*

1. Demonstrates **Traversing** through an array.
2. Demonstrates **Insertion** of elements in an array.
3. Demonstrates **Deletion** of elements from an array.
4. Performs a **Search** operation in an array.
5. **Creates** and **Traverses** a Linked List.
6. **Inserts** a node in a Linked list.
7. **Deletes** anode from a Linked list.
8. **Creates** and **Displays** aDoubly Linked list.
9. Inserts an element in a Stack using **PUSH ()**.
10. Delete an element from a Stack using **POP ()**.
11. Converts **infix** to **prefix** expression using Stacks.
12. Converts **prefix** to **postfix** expression using Stacks.
13. Calculates **factorial** of a number using Recursion**.**
14. Display the **Fibonacci Series** using Recursion.
15. **Enqueues** an element in a Queue.
16. **Dequeues** an element from a Queue.
17. Creates a **Binary search** tree.
18. A menu driven program that illustrates basic operations that can be performed on an array.

*1: Write a program that demonstrates* ***Traversing*** *through an* ***array.***

#include <stdio.h>

void traverse(int \*);

int main()

{

    // declaration

    int ar[10];

    // Input

    printf("Please enter the elements of the array here:-\n");

    for (int i = 0; i < 10; i++)

    {

        scanf("%d", &ar[i]);

    }

    // traverse/visit/access

    traverse(ar);

    return 0;

}

void traverse(int ar[])

{

    printf("The arrary is ");

    for (int i = 0; i < 10; i++)

    {

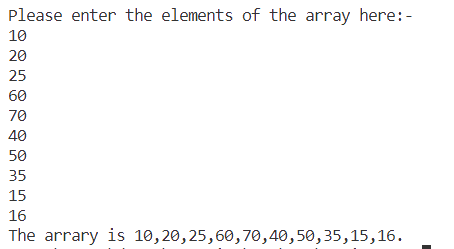
        printf("%d,", ar[i]);

    }

    printf("\b.\n");

}

*Execution:*



*2: Write a program that demonstrates* ***Insertion*** *of elements in an* ***array.***

#include <stdio.h>

int size;

void traverse(int \*);

void insertion(int \*, int, int);

int main()

{

    // declaration

    int ar[] = {1, 2, 3, 5, 6, 7}, item, index;

    size = sizeof(ar) / sizeof(ar[0]);

    // before insertion

    traverse(ar);

    // Input

    printf("Please enter the value you want to insert:-");

    scanf("%d", &item);

    printf("Please enter the index at which you want to insert:-");

    scanf("%d", &index);

    // insertion

    insertion(ar, item, index);

    // after insertion

    traverse(ar);

    return 0;

}

void traverse(int ar[])

{

    printf("The arrary is ");

    for (int i = 0; i < size; i++)

    {

        printf("%d,", ar[i]);

    }

    printf("\b.\n");

}

void insertion(int ar[], int item, int index)

{

    for (int i = size; i >= index; i--)

    {

        ar[i + 1] = ar[i];

    }

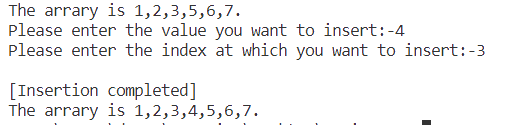
    size++;

    ar[index] = item;

    printf("\n[Insertion completed]\n");

}

*Execution:*



*3: Write a program that demonstrates* ***Deletion*** *of elements from an* ***array.***

#include <stdio.h>

int size;

void traverse(int \*);

void deletion(int \*, int);

int main()

{

    // declaration

    int ar[] = {1, 2, 3, 5, 5, 6, 7}, index;

    size = sizeof(ar) / sizeof(ar[0]);

    // before deletion

    traverse(ar);

    // Input

    printf("Please enter the location of element you want to delete:-");

    scanf("%d", &index);

    // index fixing

    index--;

    // deletion

    deletion(ar, index);

    // after deletion

    traverse(ar);

    return 0;

}

void traverse(int ar[])

{

    printf("The arrary is ");

    for (int i = 0; i < size; i++)

    {

        printf("%d,", ar[i]);

    }

    printf("\b.\n");

}

void deletion(int ar[], int index)

{

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

    {

        ar[i] = ar[i + 1];

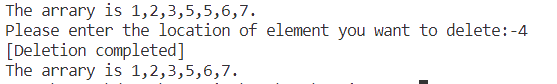
    }

    size--;

    printf("[Deletion completed]\n");

}

*Execution:*



*4: Write a program that performs a* ***Search*** *operation in an* ***array****.*

#include <stdio.h>

int size;

void traverse(int \*);

void search(int \*, int);

int main()

{ // declaration

    int ar[] = {1, 2, 3, 5, 6, 7, 8, 9, 10}, item;

    size = sizeof(ar) / sizeof(ar[0]);

    // Displaying the array

    traverse(ar);

    // Input

    printf("Please enter number you want to search here:- ");

    scanf("%d", &item);

    // Search

    search(ar, item);

    return 0;

}

void traverse(int ar[])

{

    printf("The arrary is ");

    for (int i = 0; i < size; i++)

    {

        printf("%d,", ar[i]);

    }

    printf("\b.\n");

}

// A  linear search function

void search(int ar[], int item)

{

    short flag = 0;

    for (int i = 0; i < size - 1; i++)

    { if (ar[i] == item)

        {flag = i;

         break;

        }

    }

    printf("[Search completed]\n");

    if (flag != 0)

    { printf("The element was found at an index of %d\n", flag);

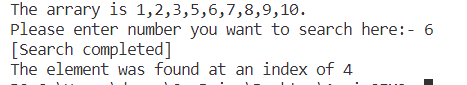
    } else

    { printf("The item was not found\n");

    }

}

*Execution:*



*5: Write a program* ***Creates*** *and* ***Traverses*** *a* ***Linked List.***

#include <stdio.h>

#include <stdlib.h>

// definition

struct node

{

    int value;

    struct node \*next;

};

void traverse(struct node \*);

int main()

{ // creation

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

    struct node \*ptr2 = (struct node \*)malloc(sizeof(struct node));

    struct node \*ptr3 = (struct node \*)malloc(sizeof(struct node));

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

    // 1st node

    head->value = 10;

    head->next = ptr2;

    // 2nd node

    ptr2->value = 20;

    ptr2->next = ptr3;

    // 3nd node

    ptr3->value = 30;

    ptr3->next = last;

    // last node

    last->value = 40;

    last->next = NULL;

    // Traversing

    traverse(head);

    return 0;

}

void traverse(struct node \*ptr)

{

    printf("The elements of the linked list are:-\n");

    while (ptr != NULL)

    {

        printf("%d,", ptr->value);

        ptr = ptr->next;

    }

    printf("\b.\n");

}

*Execution:*

**

***6*** *Write a program* ***Inserts a*** *node in a* ***Linked list****.*

#include <stdio.h>

#include <stdlib.h>

// definition

struct node

{

    int value;

    struct node \*next;

};

void traverse(struct node \*);

struct node \*insertion(struct node \*, int, int);

int main()

{ // Creation

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

    struct node \*ptr2 = (struct node \*)malloc(sizeof(struct node));

    struct node \*ptr3 = (struct node \*)malloc(sizeof(struct node));

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

    // 1st node

    head->value = 10;

    head->next = ptr2;

    // 2nd node

    ptr2->value = 20;

    ptr2->next = ptr3;

    // 3nd node

    ptr3->value = 30;

    ptr3->next = last;

    // last node

    last->value = 40;

    last->next = NULL;

    // Traversing before insetion

    traverse(head);

    // Input

    int item, index;

    printf("Please enter the value you want to insert:-");

    scanf("%d", &item);

    printf("Please enter the location at which you want to insert:-");

    scanf("%d", &index);

    index--;

    // insertion

    // Traversing after insetion

    traverse(insertion(head, item, index));

    return 0;

}

void traverse(struct node \*ptr)

{

    printf("The elements of the linked list are:-\n");

    while (ptr != NULL)

    {

        printf("%d,", ptr->value);

        ptr = ptr->next;

    }

    printf("\b.\n");

}

struct node \*insertion(struct node \*ptr, int item, int index)

{

    struct node \*head = ptr;

    // creating the node to store the value

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

    new->value = item;

    printf("\n[Insetion Complete]\n");

    // linking

    if (index <= 0)

    {

        new->next = ptr;

        return new;

    }

    else if (index >= 4)

    {

        new->next = NULL;

        while (ptr->next != NULL)

        {

            ptr = ptr->next;

        }

        ptr->next = new;

        return head;

    }

    else

    {

        int i = 1;

        while (index != i)

        {

            i++;

            ptr = ptr->next;

        }

        struct node \*temp = ptr->next;

        ptr->next = new;

        new->next = temp;

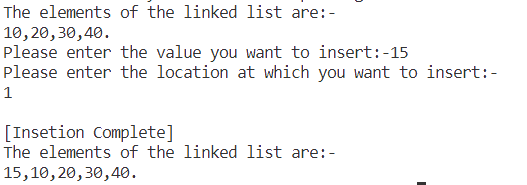
        return head;

    }

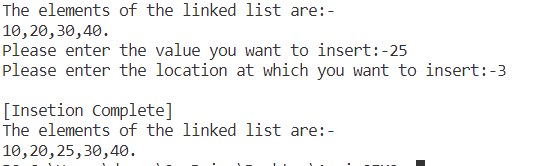
}

*Execution:*

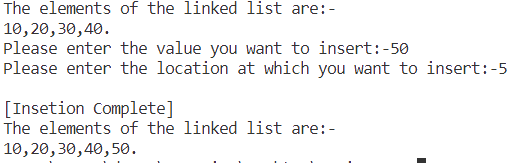
1.At Beginning

**

2.Inbetween

**

3.At last

**

***7:*** *Write a program* ***deletes*** *a node from a* ***Linked list****.*

#include <stdio.h>

#include <stdlib.h>

// definition

struct node

{

    int value;

    struct node \*next;

};

void traverse(struct node \*);

struct node \*deletion(struct node \*, int);

int main()

{ // Creation

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

    struct node \*ptr2 = (struct node \*)malloc(sizeof(struct node));

    struct node \*ptr3 = (struct node \*)malloc(sizeof(struct node));

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

    // 1st node

    head->value = 10;

    head->next = ptr2;

    // 2nd node

    ptr2->value = 20;

    ptr2->next = ptr3;

    // 3nd node

    ptr3->value = 30;

    ptr3->next = last;

    // last node

    last->value = 40;

    last->next = NULL;

    // Traversing before deletion

    traverse(head);

    // Input

    int index;

    printf("Please enter the location of element you want to delete:-");

    scanf("%d", &index);

    // index fixing

    index--;

    // deletion

    // Traversing after deletion

    traverse(deletion(head, index));

    return 0;

}

void traverse(struct node \*ptr)

{

    printf("The elements of the linked list are:-\n");

    while (ptr != NULL)

    {

        printf("%d,", ptr->value);

        ptr = ptr->next;

    }

    printf("\b.\n");

}

struct node \*deletion(struct node \*ptr, int index)

{

    struct node \*head = ptr;

    printf("\n[Deletion Complete]\n");

    if (index <= 0)

    {

        struct node \*ptr2 = ptr->next;

        free(head);

        return ptr2;

    }

    else if (index >= 3)

    {

        struct node \*del = ptr->next;

        while (del->next != NULL)

        {

            ptr = ptr->next;

            del = del->next;

        }

        ptr->next = NULL;

        free(del);

        return head;

    }

    else

    {

        int i = 0;

        while (index - 1 != i)

        {

            i++;

            ptr = ptr->next;

        }

        struct node \*del = ptr->next;

        ptr->next = del->next;

        free(del);

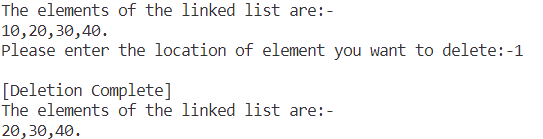
        return head;

    }

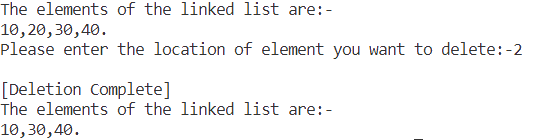
}

*Execution:*

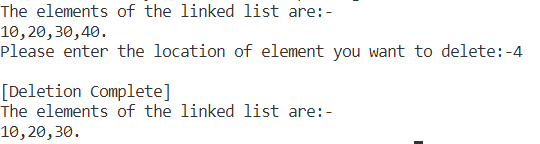
1.At Beginning



2.Inbetween



3.At last



***8:*** *Write a program* ***Creates*** *and* ***Displays*** *a* ***Double Linked list.***

#include <stdio.h>

#include <stdlib.h>

// Two pointers required

struct node

{ struct node \*prew;

    int value;

    struct node \*next;

};

void traverse(struct node \*);

int main()

{ // creation

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

    struct node \*ptr2 = (struct node \*)malloc(sizeof(struct node));

    struct node \*ptr3 = (struct node \*)malloc(sizeof(struct node));

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

    // 1st node

    head->prew = NULL;

    head->value = 10;

    head->next = ptr2;

    // 2nd node

    ptr2->prew = head;

    ptr2->value = 20;

    ptr2->next = ptr3;

    // 3nd node

    ptr3->prew = ptr2;

    ptr3->value = 30;

    ptr3->next = last;

    // last node

    last->prew = ptr3;

    last->value = 40;

    last->next = NULL;

    // Traversing

    traverse(head);

    return 0;

}

void traverse(struct node \*ptr)

{ printf("The doubly linked list is:-\n");

    while (ptr != NULL)

    {

        printf("%p[(%p)-%d-(%p)]-->  ", ptr, ptr->prew, ptr->value, ptr->next);

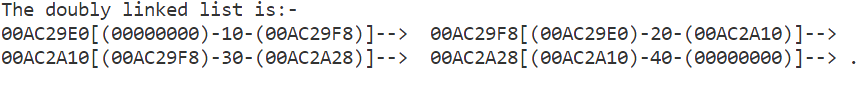
        ptr = ptr->next;

    }

    printf("\b.\n");

}

*Execution:*

**

***9:*** *Write a program* inserts an element in a **Stack** using **PUSH ()**

#include <stdio.h>

// stack declaraion

#define stack\_size 4

int stack[stack\_size];

int top = -1;

// function prototyping

void display(void);

void push(int);

// other global variables

short flag = 1;

int main()

{

    int item;

    display();

    while (flag)

    {

        char c;

        printf("Enter p to push and x to exit:- ");

        scanf("\n%c", &c);

        switch (c)

        {

        case 'p':

            printf("Please enter the number you want to push in the stack:- ");

            scanf("%d", &item);

            push(item);

            break;

        case 'x':

            flag = 0;

            break;

        }

    }

    display();

    return 0;

}

void display()

{

    if (top > -1)

    {

        printf("The elements of the stack are : -\n");

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

        {

            printf("%d, ", stack[i]);

        }

        printf("\b\b.\n");

    }

    else

    {

        printf("The stack is empty\n");

    }

}

void push(int item)

{

    if (top + 1 < stack\_size)

    {

        top++;

        stack[top] = item;

        printf("[Successfully Pushed the element]\n");

    }

    else

    {

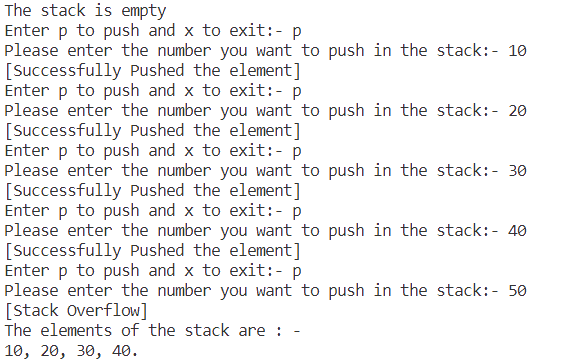
        flag = 0;

        printf("[Stack Overflow]\n");

    }

}

*Execution:*



***10:*** *Write a program deletes an element from a* ***Stack*** *using* ***Pop();***

#include <stdio.h>

// stack declaraion

#define stack\_size 4

int stack[stack\_size];

int top = -1;

// function prototyping

void display(void);

void push(int);

void pop(void);

int main()

{

    // initialization

    push(10);

    push(20);

    char c;

    display();

    printf("Please enter x to pop a element:- ");

    scanf("%c", &c);

    if (c == 'x')

    {

        pop();

        display();

    }

    return 0;

}

void display()

{

    if (top > -1)

    {

        printf("The elements of the stack are : -\n");

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

        {

            printf("%d, ", stack[i]);

        }

        printf("\b\b.\n");

    }

    else

    {

        printf("The stack is empty\n");

    }

}

void push(int item)

{

    if (top + 1 < stack\_size)

    {

        top++;

        stack[top] = item;

        printf("[Successfully Pushed the element]\n");

    }

    else

    {

        printf("[Stack Overflow]\n");

    }

}

void pop()

{

    if (top > -1)

    {

        top--;

        printf("[Successfully Popped the top element]\n");

    }

    else

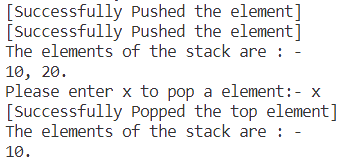
    {

        printf("[Stack Underflow]\n");

    }

}

*Execution:*

**

***11:*** *Write a program that* converts **infix** to **prefix** expression using Stacks.

#include <stdio.h>

#include <string.h>

// expresion size

#define expresion\_size 16

// stack declaraion

#define stack\_size 16

char stack[stack\_size];

int top = -1;

//  function prototyping

void push(char);

void pop(void);

char peek(void);

void in\_post(char \*);

short operator(char);

short percedence(char);

int main()

{

    char infix[expresion\_size];

    printf("Please enter the infix expresion here:- ");

    scanf("%s", infix);

    printf("The prefix statement is:- ");

    in\_post(strrev(infix));

    return 0;

}

void push(char c)

{

    if (top + 1 < stack\_size)

    {

        top++;

        stack[top] = c;

    }

    else

    {

        printf("[Stack Overflow]\n");

    }

}

void pop()

{

    if (top > -1)

    {

        top--;

    }

    else

    {

        printf("[Stack Underflow]\n");

    }

}

char peek()

{

    if (top == -1)

    {

        return 'E';

    }

    else

    {

        return stack[top];

    }

}

short check\_operator(char c)

{

    char operator[] = {'+', '\*', '-', '/'};

    for (int i = 0; i < 4; i++)

    {

        if (c == operator[i])

        {

            return 1;

        }

    }

    return 0;

}

short precedence(char a)

{

    if (a == 'E')

    {

        return -1;

    }

    int ap;

    switch (a)

    {

    case '+':

        ap = 0;

        break;

    case '-':

        ap = 0;

        break;

    case '\*':

        ap = 1;

        break;

    case '/':

        ap = 1;

    }

    return ap;

}

void in\_post(char infix[])

{

    int i = 0, p = 0;

    char postfix[expresion\_size];

    while (infix[i] != '\0')

    {

        if (check\_operator(infix[i]))

        { // minor tweek for prefix

            if (precedence(infix[i]) >= precedence(peek()))

            {

                push(infix[i]);

                i++;

            }

            else

            {

                postfix[p] = peek();

                p++;

                pop();

            }

        }

        else

        {

            postfix[p] = infix[i];

            i++;

            p++;

        }

    }

    while (1)

    {

        postfix[p] = peek();

        p++;

        if (top == 0)

        {

            break;

        }

        pop();

    }

    postfix[p] = '\0';

    printf("%s,", strrev(postfix));

}

*Execution:*

**

**

**

***12:*** *Write a program that* converts **prefix** to **postfix** expression using Stacks.

#include <stdio.h>

#include <string.h>

// expresion size

#define expresion\_size 16

// stack declaraion

#define stack\_size 16

char stack[stack\_size][stack\_size];

int top = -1;

//  function prototyping

void display();

void push(char \*);

void pop(void);

char \*peek(void);

void pre\_post(char \*);

short operator(char);

int main()

{

    char prefix[expresion\_size];

    printf("Please enter the prefix expresion here:- ");

    scanf("%s", prefix);

    printf("The postfix statement is:- ");

    pre\_post(strrev(prefix));

    return 0;

}

void push(char \*c)

{

    if (top + 1 < stack\_size)

    {

        top++;

        strcpy(stack[top], c);

    }

    else

    {

        printf("[Stack Overflow]\n");

    }

}

void pop()

{

    if (top > -1)

    {

        top--;

    }

    else

    {

        printf("[Stack Underflow]\n");

    }

}

char \*peek()

{

    if (top == -1)

    {

        return "Empty";

    }

    else

    {

        return stack[top];

    }

}

short check\_operator(char c)

{

    char operator[] = {'+', '\*', '-', '/'};

    for (int i = 0; i < 4; i++)

    {

        if (c == operator[i])

        {

            return 1;

        }

    }

    return 0;

}

void pre\_post(char prefix[])

{

    int pr = 0;

    char a[16];

    char b[16];

    while (prefix[pr] != '\0')

    {

        char str[2] = {prefix[pr], '\0'};

        if (!check\_operator(prefix[pr]))

        {

            push(str);

        }

        else

        {

            strcpy(a, peek());

            pop();

            strcpy(b, peek());

            pop();

            strcat(a, b);

            strcat(a, str);

            push(a);

        }

        pr++;

    }

    display();

}

void display()

{

    if (top > -1)

    {

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

        {

            printf("%s", stack[i]);

        }

    }

    else

    {

        printf("The stack is empty\n");

    }

}

*Execution:*







***13:*** *Write a program that calculates* ***factorial*** *of a number using Recursion****.***

#include <stdio.h>

int factorial(int);

int main()

{

    int num;

    printf("Please enter the number you want the factorial of : - ");

    scanf("%d", &num);

    printf("the factorial of the number is %d", factorial(num));

    return 0;

}

int factorial(int num)

{

    if (num == 0)

    {

        return 1;

    }

    return num \* factorial(num - 1);

}

*Execution:*







***14:*** *Write a program that display the* ***Fibonacci Series*** *using Recursion.*

#include <stdio.h>

int x = 0, y = 1, z;

int fabonacci(int);

int main()

{

    int limit;

    printf("Please enter the limit of elements you want here:- ");

    scanf("%d", &limit);

    printf("The Fabonacci series is :- ");

    for (int i = 0; i < limit; i++)

    {

        printf("%d,", fabonacci(i));

    }

    printf("\b.\n");

    return 0;

}

int fabonacci(int i)

{

    if (i == 0 || i == 1)

    {

        return i;

    }

    else

    {

        return fabonacci(i - 1) + fabonacci(i - 2);

    }

}

*Execution:*



***15:*** *Write a program that* ***Enqueues*** *an element in a Queue.*

#include <stdio.h>

// queue declaration

#define queue\_size 10

int queue[queue\_size];

int front = 0;

int rare = 0;

// function prototyping

void display();

void enqueue(int);

int main()

{

    int item;

    display();

    printf("Enter the element you want to enqueue(insert): -");

    scanf("%d", &item);

    enqueue(item);

    display();

}

void display()

{

    if (front == rare)

    { printf("[Queue is Empty]\n");

    }

    else

    {printf("The queue is :- \n");

        for (int i = front; i < rare; i++)

        {

            printf("%d<-", queue[i]);

        }

    }

}

void enqueue(int item)

{

    if (rare == queue\_size)

    { printf("[overflow]\n");

    }

    else

    {printf("[Enqueueing Successful]\n");

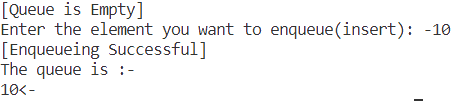
        queue[rare] = item;

        rare++;

    }

}

*Execution:*

**

***16:*** *Write a program that* ***Dequeues*** *an element from a Queue.*

#include <stdio.h>

// queue declaration

#define queue\_size 10

int queue[queue\_size];

int front = 0;

int rare = 0;

// function prototyping

void display();

void dequeue();

void enqueue(int);

int main()

{

    char c;

    enqueue(10);

    enqueue(20);

    enqueue(40);

    display();

    printf("\nPress Enter to dequeue-");

    scanf("%c", &c);

    dequeue();

    display();

}

void display()

{

    if (front == rare)

    {

        printf("[Queue is Empty]\n");

    }

    else

    {

        printf("The queue is :- \n");

        for (int i = front; i < rare; i++)

        {

            printf("%d<-", queue[i]);

        }

    }

}

void enqueue(int item)

{

    if (rare == queue\_size)

    {

        printf("[overflow]\n");

    }

    else

    {

        printf("[Enqueueing Successful]\n");

        queue[rare] = item;

        rare++;

    }

}

void dequeue()

{

    if (rare == 0)

    {

        printf("[Queue is Empty]\n");

    }

    else

    {

        printf("[Dequeueing Successful]\n");

        for (int i = front; i < rare; i++)

        {

            queue[i] = queue[i + 1];

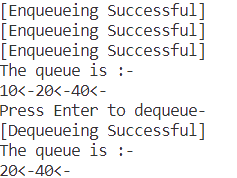
        }

        rare--;

    }

}

*Execution:*

**