1
Element does not exist...!
1.INSERT 2.DELETE 3.FIND 4.PRINT 5.QUIT
Enter the choice:
4
The list element are: 54 ->
1.INSERT 2.DELETE 3.FIND 4.PRINT 5.QUIT
Enter the choice:
1
Enter the element to be inserted:
65
Enter the position of the element:
2
1.INSERT 2.DELETE 3.FIND 4.PRINT 5.QUIT
Enter the choice:
4
The list element are: 54 -> 65 ->
1.INSERT 2.DELETE 3.FIND 4.PRINT 5.QUIT
Enter the choice:
5

S.No: 16	Exp. Name: ***Implementation of Circular Queue using Dynamic Array***	**Date: 2023-06-11**

Aim:

Write a program to implement `circular queue` using **dynamic array**.

```
Sample Input and Output:
    Enter the maximum size of the circular queue : 3
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Circular queue is underflow.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 111
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 222
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 333
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 444
    Circular queue is overflow.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Elements in the circular queue : 111 222 333
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 111
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 444
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Elements in the circular queue : 222 333 444
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 222
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 333
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 444
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 4
```

Source Code:

CQueueUsingDynamicArray.c

```
#include <stdio.h>
#include <stdlib.h>
int *cqueue;
int front, rear;
int maxSize;
void initCircularQueue()
{
        cqueue = (int *)malloc(maxSize * sizeof(int));
        front = -1;rear = -1;
}
void dequeue()
{
        if (front == -1)
        {
                printf("Circular queue is underflow.\n");
        }
        else
        {
                printf("Deleted element = %d\n", *(cqueue + front));
                if (rear == front)
                {
                        rear = front = -1;
                }
                else if
                (front == maxSize - 1)
                {
                        front = 0;
                }
                else
                {
                        front++;
                }
        }
}
void enqueue(int x)
{
        if (((rear == maxSize - 1) && (front == 0)) || (rear + 1 == front))
        {
                printf("Circular queue is overflow.\n");
        }
        else
        {
                if (rear == maxSize - 1)
                {
                        rear = -1;
                }
                else if (front == -1)
                {
                        front = 0;
                }
                rear++;
                cqueue[rear] = x;
                printf("Successfully inserted.\n");
        }
}
```

```
        int i;
        if (front == -1 && rear == -1)
        {
                printf("Circular queue is empty.\n");
        }
        else
        {
                printf("Elements in the circular queue : ");
                if (front <= rear){for (i = front; i <= rear; i++)
                {
                        printf("%d ", *(cqueue + i));
                }
        }
        else
        {
                for (i = front; i <= maxSize - 1; i++)
                {
                        printf("%d ", *(cqueue + i));
                }
                for (i = 0; i <= rear; i++)
                {
                        printf("%d ", *(cqueue + i));
                }
        }
        printf("\n");
        }
}
int main()
{
        int op, x;
        printf("Enter the maximum size of the circular queue : ");
        scanf("%d", &maxSize);
        initCircularQueue();
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:printf("Enter element : ");
                        scanf("%d",&x);
                        enqueue(x);
                        break;
                        case 2:dequeue();break;
                        case 3:display();break;
                        case 4:exit(0);
                }
        }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output
Enter the maximum size of the circular queue :
3
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Circular queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
111
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
222
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
333
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
444
Circular queue is overflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the circular queue : 111 222 333
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 111
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
444
Successfully inserted.

1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the circular queue : 222 333 444
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 222
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 333
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 444
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
4

S.No: 17	Exp. Name: ***Write a C program to implement different Operations on Stack using Array representation***	Date: 2023-06-12

Aim:

Write a program to implement `stack` using **arrays**.

```
Sample Input and Output:
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Stack is underflow.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Stack is empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Stack is underflow.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 25
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 26
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 26 25
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 26
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is not empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Peek value = 25
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 6
```

Source Code:

StackUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
int arr[STACK_MAX_SIZE];
int top = -1;
void push(int element)
{
        if(top == STACK_MAX_SIZE - 1)
        {
                printf("Stack is overflow.\n");

        }
        else
        {
                top = top + 1;
                arr[top] = element;
                printf("Successfully pushed.\n");

        }

}
void display(){
        if (top < 0)
        {
                printf("Stack is empty.\n");

        }
        else
        {
                printf("Elements of the stack are : " );
                for(int i = top; i >= 0; i--)
                {
                        printf("%d ", arr[i]);

                }
                printf("\n");

        }

}
void pop(){
        int x;
        if(top < 0){
                printf("Stack is underflow.\n");

        }
        else
        {
                x = arr[top];
                top = top - 1;
                printf("Popped value = %d\n",x);

        }

}
```

```
        if(top < 0)
        {
                printf("Stack is underflow.\n");

        }
        else
        {
                x = arr[top];
                printf("Peek value = %d\n",x);

        }

}
void isEmpty(){
        if (top < 0){
                printf("Stack is empty.\n");

        }
        else
        {
                printf("Stack is not empty.\n");

        }

}
int main(){
        int op, x;
        while(1){
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op){
                    case 1:
                          printf("Enter element : ");
                          scanf("%d", &x);push(x);
                          break;
                        case 2:
                          pop();
                          break;
                        case 3:
                          display();
                          break;
                        case 4:
                          isEmpty();
                          break;
                        case 5:
                          peek();
                          break;
                        case 6:
                          exit(0);

                }

        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
10
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
20
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
30
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 30 20 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 30
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 30
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 20
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :

4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 18	Exp. Name: ***Write a C program to implement different Operations on Stack using Linked Lists***	Date: 2023-06-12

Aim:

Write a program to implement stack using **linked lists**.

```
Sample Input and Output:
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 33
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 22
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 55
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 66
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 66 55 22 33
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 66
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 55
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 22 33
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Peek value = 22
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is not empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 6
```

Source Code:

StackUsingLList.c

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

};
typedef struct stack *stk;
stk top = NULL;
stk push(int x)
{
        stk temp;
        temp = (stk)malloc(sizeof(struct stack));
        if(temp == NULL)
        {
                printf("Stack is overflow.\n");

        }
        else
        {
                temp -> data = x;
                temp -> next = top;
                top = temp;
                printf("Successfully pushed.\n");

        }

}
void display()
{
        stk temp = top;
        if(temp == NULL)
        {
                printf("Stack is empty.\n");
                }
                else
                {
                        printf("Elements of the stack are : ");
                        while(temp != NULL)
                        {
                                printf("%d ", temp -> data);
                                temp = temp -> next;

                        }
                        printf("\n");

                }

}
stk pop()
{
        stk temp;
        if(top == NULL)
        {
```

```
        }
        else
        {
                temp = top;
                top = top -> next;
                printf("Popped value = %d\n", temp -> data);
                free(temp);

        }

}
void peek()
{
        stk temp;
        if(top == NULL)
        {
                printf("Stack is underflow.\n");

        }
        else
        {
                temp = top;
                printf("Peek value = %d\n", temp -> data);

        }

}
void isEmpty()
{
        if(top == NULL)
        {
                printf("Stack is empty.\n");

        }
        else
        {
                printf("Stack is not empty.\n");

        }

}
int main()
{
        int op, x;
        while(1)
        {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op)
                {
                        case 1: printf("Enter element : ");
                        scanf("%d", &x);
                        push(x);
                        break;
```

```
                        break;
                        case 3:
                        display();
                        break;
                        case 4:
                        isEmpty();
                        break;
                        case 5:
                        peek();
                        break;
                        case 6:
                        exit(0);

                }

        }

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
33
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
22
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
55
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
66
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :

3
Elements of the stack are : 66 55 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 66
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 55
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 22
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :

23
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
24
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 24 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 19	Exp. Name: ***Write a C program to implement different Operations on Queue using Array representation***	Date: 2023-06-12

Aim:

Write a program to implement queue using **arrays**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 23
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 56
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the queue : 23 56
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 4
    Queue is not empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 23
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 56
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 4
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

QUsingArray.c

```
#include <conio.h>
#include <stdio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int x)
{
        if (rear == MAX - 1)
        {
                printf("Queue is overflow.\n");

        }
        else
        {
                rear++;
                queue[rear] = x;
                printf("Successfully inserted.\n");

        }
        if (front == -1)
        {
                front++;

        }

}
void dequeue()
{
        if (front == -1)
        {
                printf("Queue is underflow.\n");

        }
        else
        {
                printf("Deleted element = %d\n",queue[front]);
                if (rear == front)
                {
                        rear = front = -1;

                }
                else
                {
                        front++;

                }

        }

}
void display()
{
        if (front == -1 && rear == -1)
        {
                printf("Queue is empty.\n");
```

```
        else
        {
                printf("Elements in the queue : ");
                for (int i = front; i <= rear; i++)
                {
                        printf("%d ",queue[i]);

                }
                printf("\n");

        }

}
void size()
{
        if(front == -1 && rear == -1)
        printf("Queue size : 0\n");
        else
        printf("Queue size : %d\n",rear-front+1);

}
void isEmpty()
{
        if(front == -1 && rear == -1)
        printf("Queue is empty.\n");
        else
        printf("Queue is not empty.\n");

}
int main()
{
        int op, x;
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                        printf("Enter element : ");
                        scanf("%d",&x);
                        enqueue(x);
                        break;
                        case 2:
                        dequeue();
                        break;
                        case 3:
                        display();
                        break;
                        case 4:
                        isEmpty();
                        break;
                        case 5:
                        size();
```

```
                    exit(0);

                }

        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
14
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
78
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
53
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit

Enter your option :
3
Elements in the queue : 14 78 53
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 25
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
65
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2

Deleted element = 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
63
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

S.No: 20	Exp. Name: ***Write a C program to implement different Operations on Queue using Dynamic Array***	Date: 2023-06-12

ID: 224G1A05B1

Srinivasa Ramanujan Institute of Technology

Aim:

Write a program to implement queue using **dynamic array**.

In this queue implementation has

1. a pointer 'queue' to a dynamically allocated array (used to hold the contents of the queue)
2. an integer 'maxSize' that holds the size of this array (i.e the maximum number of data that can be held in this array)
3. an integer 'front' which stores the array index of the first element in the queue
4. an integer 'rear' which stores the array index of the last element in the queue.

```
Sample Input and Output:
    Enter the maximum size of the queue : 3
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Queue is underflow.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 15
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 16
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 17
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 1
    Enter element : 18
    Queue is overflow.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Elements in the queue : 15 16 17
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 15
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 16
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Elements in the queue : 17
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Deleted element = 17
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 3
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 2
    Queue is underflow.
    1.Enqueue 2.Dequeue 3.Display 4.Exit
    Enter your option : 4
```

Source Code:

QUsingDynamicArray.c

```
#include <conio.h>
#include <stdio.h>
int *queue;
int front, rear;
int maxSize;
void initQueue()
{
        queue = (int *)malloc(maxSize*sizeof(int));
        front = -1;rear = -1;

}
void enqueue(int x)
{
        if (rear == maxSize - 1)
        {
                printf("Queue is overflow.\n");

        }
        else
        {
                rear++;
                queue[rear] = x;
                printf("Successfully inserted.\n");

        }
        if (front == -1)
        {
                front++;

        }

}
void dequeue()
{
        if (front == -1)
        {
                printf("Queue is underflow.\n");

        }
        else
        {
                printf("Deleted element = %d\n", *(queue+front));
                if (rear == front)
                {
                        rear = front = -1;

                }
                else
                {
                        front++;

                }

        }
```

```
{
        if (front == -1 && rear == -1)
        {
                printf("Queue is empty.\n");

        }
        else
        {
                printf("Elements in the queue : ");
                for (int i = front; i <= rear; i++)
                {
                        printf("%d ",*(queue+i));

                }
                printf("\n");

        }

}
int main()
{
        int op, x;
        printf("Enter the maximum size of the queue : ");
        scanf("%d", &maxSize);
        initQueue();
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                        printf("Enter element : ");
                        scanf("%d",&x);
                        enqueue(x);
                        break;
                        case 2:
                        dequeue();
                        break;
                        case 3:
                        display();
                        break;
                        case 4:
                        exit(0);

                }

        }

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the maximum size of the queue :
3
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
15
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
16
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
17
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
18
Queue is overflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the queue : 15 16 17
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 15
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 16

1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the queue : 17
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 17
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
4

Test Case - 2
User Output
Enter the maximum size of the queue :
2
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
34
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
56
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
45
Queue is overflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the queue : 34 56
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :

2
Deleted element = 34
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Deleted element = 56
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
1
Enter element :
56
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
3
Elements in the queue : 56
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your option :
4

S.No: 21	Exp. Name: ***Write a C program to implement different Operations on Queue using Linked Lists***	Date: 2023-06-12

Aim:

Write a program to implement queue using **linked lists**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 57
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 87
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the queue : 57 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 57
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

QUsingLL.c

```
#include <conio.h>
#include <stdio.h>
struct queue
{
        int data;
        struct queue *next;

};
typedef struct queue *Q;
Q front = NULL, rear = NULL;
void enqueue(int element)
{
        Q temp = NULL;
        temp = (Q)malloc(sizeof(struct queue));
        if(temp == NULL)
        {
                printf("Queue is overflow.\n");

        }
        else
        {
                temp -> data = element;
                temp -> next = NULL;
                if(front == NULL)
                {
                        front = temp;

                }
                else
                {
                        rear -> next = temp;

                }
                rear = temp;
                printf("Successfully inserted.\n");

        }

}
void dequeue()
{
        Q temp = NULL;
        if(front == NULL)
        {
                printf("Queue is underflow.\n");

        }
        else
        {
                temp = front;
                if (front == rear)
                {
                        front = rear = NULL;

                }
```

```
                        front = front -> next;

                }
                printf("Deleted value = %d\n", temp -> data);
                free(temp);

        }

}
void display()
{
        if(front == NULL)
        {
                printf("Queue is empty.\n");

        }
        else
        {
                Q temp = front;
                printf("Elements in the queue : ");
                while(temp != NULL)
                {
                        printf("%d ", temp -> data);
                        temp = temp -> next;

                }
                printf("\n");

        }

}
void size()
{
        int count =0;
        if(front == NULL)
        {
                printf("Queue size : 0\n");
                }
                else
                {
                        Q temp = front;
                        while(temp != NULL)
                        {
                                temp = temp -> next;
                                count = count + 1;

                        }
                        printf("Queue size : %d\n",count);

                }

}
void isEmpty()
{
        if(front == NULL )
```

```
        }
        else
        {
                printf("Queue is not empty.\n");

        }

}
int main()
{
        int op, x;
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                        printf("Enter element : ");
                        scanf("%d",&x);
                        enqueue(x);
                        break;
                        case 2:
                        dequeue();
                        break;
                        case 3:
                        display();
                        break;
                        case 4:
                        isEmpty();
                        break;
                        case 5:
                        size();
                        break;
                        case 6:
                        exit(0);

                }

        }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.

```
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
44
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
55
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
66
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
67
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 44 55 66 67
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 44
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 55
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
```

Enter your option :
5
Queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
23
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
234
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
45
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
456
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 23
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 234 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2

Deleted value = 234
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

S.No: 22	Exp. Name: ***Reversing the links of a linked list***	Date: 2023-06-12

Aim:

Write a C program to reverse the links (not just displaying) of a linked list.
Note: Add node at the beginning.

Source Code:

reverseLinkedList.c

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node* next;
        };
        static void reverse(struct Node** head_ref)
        {
                struct Node* prev = NULL;
                struct Node* current = *head_ref;
                struct Node* next = NULL;
                while (current != NULL)
                {
                        next = current->next;
                        current->next = prev;
                        prev = current;
                        current = next;
                        }
                        *head_ref = prev;
                        }
                        void push(struct Node** head_ref, int new_data)
                        {
                                struct Node* new_node = (struct Node*) malloc(sizeof(struct
Node));
                                new_node->data = new_data;
                                new_node->next = (*head_ref);
                                (*head_ref) = new_node;
                                }
                                void printList(struct Node* head)
                                {
                                        struct Node* temp = head;
                                        while (temp != NULL)
                                        {
                                                printf("%d", temp->data);
                                                if ( temp -> next != NULL)
                                                {
                                                        printf("->");
                                                        }
                                                        temp = temp->next;
                                                        }
                                                        }
                                                        int main()
                                                        {
                                                                struct Node* head = NULL;
                                                                int i, count = 0, num = 0;
                                                                printf("How many  numbers
you want to enter:");
                                                                scanf(" %d", &count);
                                                                for (i = 0; i < count; i++)
                                                                {
                                                                        printf("Enter number
%d:", i+1);
                                                                        scanf(" %d", &num);
push(&head, num);
```

```
list:");
                                                                printList(head);
                                                                reverse(&head);
                                                                printf("\nReversed
linked list:");
                                                                printList(head);

                                                        }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
How many numbers you want to enter:
4
Enter number 1:
6
Enter number 2:
1
Enter number 3:
8
Enter number 4:
5
Given linked list:5->8->1->6
Reversed linked list:6->1->8->5

Test Case - 2
User Output
How many numbers you want to enter:
2
Enter number 1:
5
Enter number 2:
9
Given linked list:9->5
Reversed linked list:5->9

S.No: 23	Exp. Name: ***Program to insert into BST and traversal using In-order, Pre-order and Post-order***	Date: 2023-06-18

Aim:

Write a program to create a binary search tree of integers and perform the following operations using linked list.

5. Insert a node
6. In-order traversal
7. Pre-order traversal
8. Post-order traversal

Source Code:

BinarySearchTree.c

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

};
typedef struct node *BSTNODE;
BSTNODE newNodeInBST(int item)
{
        BSTNODE temp = (BSTNODE)malloc(sizeof(struct node));
        temp->data = item;
        temp->left = temp->right = NULL;
        return temp;

}
void inorderInBST(BSTNODE root)
{
        if (root != NULL)
        {
                inorderInBST(root->left);
                printf("%d ", root->data);
                inorderInBST(root->right);

        }

}
void preorderInBST(BSTNODE root)
{
        if (root != NULL)
        {
                printf("%d ", root->data);
                preorderInBST(root->left);
                preorderInBST(root->right);
        }

}
void postorderInBST(BSTNODE root)
{
        if (root != NULL)
        {
                postorderInBST(root->left);
                postorderInBST(root->right);
                printf("%d ", root->data);
        }

}
BSTNODE insertNodeInBST(BSTNODE node, int ele)
{
        if (node == NULL)
        {
                printf("Successfully inserted.\n");
                return newNodeInBST(ele);
```

```
        node->left = insertNodeInBST(node->left,ele);
        else if (ele > node->data)
        node->right = insertNodeInBST(node->right,ele);
        else
        printf("Element already exists in BST.\n");
        return node;

}
void main()
{
        int x, op;
        BSTNODE root = NULL;
        while(1)
        {
                printf("1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder
Traversal 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op)
                {
                        case 1:
                        printf("Enter an element to be inserted : ");
                        scanf("%d", &x);
                        root = insertNodeInBST(root,x);
                        break;
                        case 2:
                        if(root == NULL)
                        {
                                printf("Binary Search Tree is empty.\n");

                        }
                        else
                        {
                                printf("Elements of the BST (in-order traversal): ");
                                inorderInBST(root);
                                printf("\n");

                        }
                        break;
                        case 3:
                        if(root == NULL)
                        {
                                printf("Binary Search Tree is empty.\n");

                        }
                        else
                        {
                                printf("Elements of the BST (pre-order traversal): ");
                                preorderInBST(root);
                                printf("\n");

                        }
                        break;
                        case 4:
                        if(root == NULL)
```

```
                        }
                        else
                        {
                                printf("Elements of the BST (post-order traversal): ");
                                postorderInBST(root);
                                printf("\n");

                        }
                        break;
                        case 5:
                        exit(0);

                }

        }

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
100
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
20
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
200
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
10
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :

1
Enter an element to be inserted :
30
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
150
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
300
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
2
Elements of the BST (in-order traversal): 10 20 30 100 150 200 300
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
3
Elements of the BST (pre-order traversal): 100 20 10 30 200 150 300
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
4
Elements of the BST (post-order traversal): 10 30 20 150 300 200 100
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
5

Test Case - 2
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
25
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
63
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :

1
Enter an element to be inserted :
89
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
45
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
65
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
28
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
4
Elements of the BST (post-order traversal): 28 45 65 89 63 25
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
3
Elements of the BST (pre-order traversal): 25 63 45 28 89 65
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
2
Elements of the BST (in-order traversal): 25 28 45 63 65 89
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
5

S.No: 24	Exp. Name: ***Write a Program to Search an element using Binary Search and Recursion***	Date: 2023-06-18

ID: 224G1A05B1

2022-2026-CSE-B

Srinivasa Ramanujan Institute of Technology

Aim:

Write a program to `search` the given element from a list of elements with `binary search` technique using **recursion**.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 5
```

Next, the program should print the following messages one by one on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 33 55 22 44 11
```

then the program should **print** the result as:

```
After sorting the elements are : 11 22 33 44 55
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 11
```

then the program should **print** the result as:

```
The given key element 11 is found at position : 0
```

Similarly, if the key element is given as `18` for the above example then the program should print the output as:

```
The given key element 18 is not found
```

Note: Write the functions **read()**, **bubbleSort()**, **display()** and **binarySearch()** in `BinarySearch.c`

Source Code:

`BinarySearch.c`

```
#include <stdio.h>
void read(int a[20], int n)
{
        int i;
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++)
        {
                scanf("%d", &a[i]);

        }

}
void bubbleSort(int a[20], int n)
{
        int i, j, temp;
        for (i = 0; i < n - 1; 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;

                        }

                }

        }

}
void display(int a[20], int n)
{
        int i;
        for (i = 0; i < n; i++)
        {
                printf("%d ", a[i]);

        }
        printf("\n");

}
int binarySearch(int a[20], int low, int high, int key)
{
        int mid;
        if (low <= high)
        {
                mid = (low + high) / 2;
                if (a[mid] == key)
                return mid;
                else if (key < a[mid])binarySearch(a, low, mid - 1, key);
                else if (key > a[mid])binarySearch(a, mid + 1, high, key);

        }
```

```
                return -1;

        }

}
void main()
{
        int a[20], n, key, flag;
        printf("Enter value of n : ");
        scanf("%d", &n);
        read(a, n);
        bubbleSort(a, n);
        printf("After sorting the elements are : ");
        display(a, n);
        printf("Enter key element : ");
        scanf("%d", &key);
        flag = binarySearch(a, 0, n - 1, key);
        if (flag == -1)
        {
                printf("The given key element %d is not found\n", key);

        }
        else
        {
                printf("The given key element %d is found at position : %d\n", key, flag);

        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n :
5
Enter 5 elements :
33 55 22 44 11
After sorting the elements are : 11 22 33 44 55
Enter key element :
11
The given key element 11 is found at position : 0

Test Case - 2
User Output
Enter value of n :
4
Enter 4 elements :
23 9 45 18

```
After sorting the elements are : 9 18 23 45
Enter key element :
24
The given key element 24 is not found
```

S.No: 25	Exp. Name: ***Graph traversals implementation - Breadth First Search***	Date: 2023-06-18

Aim:

Write a program to implement Breadth First Search of a graph.

Source Code:

GraphsBFS.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 99
struct node
{
        struct node *next;
        int vertex;

};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int queue[MAX], front = -1,rear = -1;
int n;
void insertQueue(int vertex)
{
        if(rear == MAX-1)
        printf("Queue Overflow.\n");
        else
        {
                if(front == -1)
                front = 0;
                rear = rear+1;
                queue[rear] = vertex ;

        }

}
int isEmptyQueue()
{
        if(front == -1 || front > rear)
        return 1;
        else
        return 0;

}
int deleteQueue()
{
        int deleteItem;
        if(front == -1 || front > rear)
        {
                printf("Queue Underflow\n");
                exit(1);

        }
        deleteItem = queue[front];
        front = front+1;
        return deleteItem;

}
void BFS(int v)
{
        int w;
        insertQueue(v);
        while(!isEmptyQueue())
```

```
                printf("\n%d",v);
                visited[v]=1;
                GNODE g = graph[v];
                for(;g!=NULL;g=g->next)
                {
                        w=g->vertex;
                        if(visited[w]==0)
                        {
                                insertQueue(w);visited[w]=1;

                        }

                }

        }

}
void main()
{
        int N, E, s, d, i, j, v;
        GNODE p, q;
        printf("Enter the number of vertices : ");
        scanf("%d",&N);printf("Enter the number of edges : ");
        scanf("%d",&E);
        for(i=1;i<=E;i++)
        {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                q=(GNODE)malloc(sizeof(struct node));
                q->vertex=d;q->next=NULL;
                if(graph[s]==NULL)
                {
                        graph[s]=q;

                }
                else
                {
                        p=graph[s];
                        while(p->next!=NULL)
                        p=p->next;p->next=q;

                }

        }
        for(i=1;i<=n;i++)
        visited[i]=0;
        printf("Enter Start Vertex for BFS : ");
        scanf("%d", &v);printf("BFS of graph : ");
        BFS(v);
        printf("\n");

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
5
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter source :
1
Enter destination :
4
Enter source :
4
Enter destination :
2
Enter source :
2
Enter destination :
3
Enter source :
4
Enter destination :
5
Enter Start Vertex for BFS :
1
BFS of graph :
1
2
4
3
5

Test Case - 2
User Output
Enter the number of vertices :
4
Enter the number of edges :
3
Enter source :
1
Enter destination :

2
Enter source :
2
Enter destination :
3
Enter source :
3
Enter destination :
4
Enter Start Vertex for BFS :
2
BFS of graph :
2
3
4

S.No: 26	Exp. Name: ***Graph traversals implementation - Depth First Search***	Date: 2023-06-18

Aim:

Write a program to implement Depth First Search for a graph.

Source Code:

GraphsDFS.c