

QUEUE

```
/*
*****
* QUEUE IMPLEMENTATION IN C *
*****
*/
#include <stdio.h>
#include <conio.h>
#define MAX 10

int queue[MAX], front = -1, rear = -1;
struct node {
    int data;
    struct node * link;
} * front_, * rear_;

struct pq {
    int data;
    int priority;
    struct pq * link;
} * pq_front;

void static_insert() {
    int data;
    if (rear == MAX - 1) {
        printf("QUEUE OVERFLOW!");
        return;
    }
    printf("ENTER DATA : ");
    scanf("%d", & data);
    if (rear == -1) {
        rear = front = 0;
    } else rear++;
    queue[rear] = data;
}

void static_delete() {
    int data;
    if (front == -1) {
        printf("QUEUE IS EMPTY!");
    }
}
```

```
        return;
    }
    data = queue[front];
    if (front == rear) {
        front = rear = -1;
    } else {
        front++;
    }
    printf("DELETED ELEMENT IS : %d", data);
}

void static_display() {
    int i;
    if (front == -1) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    printf("QUEUE : ");
    for (i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
}

void dynamic_insert() {
    struct node * temp;
    int data;
    printf("ENTER DATA : ");
    scanf("%d", & data);
    temp = (struct node * ) malloc(sizeof(struct node));
    temp->link = NULL;
    temp->data = data;
    if (rear_ == NULL) {
        rear_ = front_ = temp;
    } else {
        rear_->link = temp;
        rear_ = temp;
    }
}

void dynamic_delete() {
    struct node * temp;
    if (front_ == NULL) {
        printf("QUEUE IS EMPTY!");
    }
}
```

```
        return;
    }
    temp = front_;
    printf("DELETED ELEMENT IS : %d", front_>data);
    front_ = front_>link;
    free(temp);
}

void dynamic_display() {
    struct node * temp;
    if (front_ == NULL) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    temp = front_;
    printf("QUEUE : ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
}

void circular_insert() {
    int data;
    if ((front == (rear + 1) % MAX)) {
        printf("QUEUE OVERFLOW!");
        return;
    }
    printf("ENTER DATA : ");
    scanf("%d", & data);
    if (rear == -1) {
        rear = front = 0;
    } else {
        rear = (rear + 1) % MAX;
    }
    queue[rear] = data;
}

void circular_delete() {
    int data;
    if (front == -1) {
        printf("QUEUE UNDERFLOW!");
        return;
    }
}
```

```
data = queue[front];
if (front == rear) {
    front = rear = -1;
} else {
    front = (front + 1) % MAX;
}
printf("DELETED ELEMENT IS : %d", data);
}

void circular_display() {
    int i;
    if (front == -1) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    printf("QUEUE : ");
    if (front <= rear) {
        for (i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    } else {
        for (i = front; i < MAX; i++) {
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    }
}

void double_insert() {
    int ch, data, i;
    printf("1. INSERT AT FRONT END\n");
    printf("2. INSERT AT REAR END\n");
    printf("ENTER YOUR CHOICE : ");
    scanf("%d", & ch);
    switch (ch) {
    case 1:
        if (front == 0 && rear == MAX - 1) {
            printf("QUEUE OVERFLOW!");
            return;
        }
        printf("ENTER DATA : ");
```

```
scanf("%d", & data);
if (front == -1) {
    front = rear = 0;
} else if (front == 0 && rear != MAX - 1) {
    for (i = rear + 1; i > 0; i--) {
        queue[i] = queue[i - 1];
    }
    rear++;
} else {
    front--;
}
queue[front] = data;
break;
case 2:
    if (rear == MAX - 1) {
        printf("QUEUE OVERFLOW!");
        return;
    }
    printf("ENTER DATA : ");
    scanf("%d", & data);
    if (rear == -1) {
        rear = front = 0;
    } else if (front != 0 && rear == MAX - 1) {
        for (i = front - 1; i < rear; i++) {
            queue[i] = queue[i + 1];
        }
        front--;
    } else {
        rear++;
    }
    queue[rear] = data;
    break;
default:
    printf("WRONG CHOICE! ");
}
}

void double_delete() {
    int ch, data;
    if (front == -1 || rear == -1) {
        printf("QUEUE IS EMPTY!");
        return;
    }
}
```

```
printf("1. DELETE AT FRONT END\n");
printf("2. DELETE AT REAR END\n");
printf("ENTER YOUR CHOICE : ");
scanf("%d", & ch);
switch (ch) {
    case 1:
        data = queue[front];
        if (front == rear) {
            rear = front = -1;
        } else {
            front++;
        }
        printf("DELETE ELEMENT IS : %d", data);
        break;
    case 2:
        data = queue[rear];
        if (rear == front) {
            rear = front = -1;
        } else {
            rear--;
        }
        printf("DELETED ELEEEMENT IS : %d", data);
        break;
    default:
        printf("WRONG CHOICE! ");
}

void double_display() {
    int i;
    if (front == -1 || rear == -1) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    printf("QUEUE : ");
    for (i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
}

void pq_insert() {
    struct pq * temp, * temp2;
    int data, priority;
    printf("ENTER DATA : ");
```

```
scanf("%d", & data);
printf("ENTER PRIORITY : ");
scanf("%d", & priority);
temp = (struct pq * ) malloc(sizeof(struct pq));
temp->data = data;
temp->priority = priority;
temp->link = NULL;
if (pq_front==NULL||pq_front->priority > temp->priority) {
    temp-> link = pq_front;
    pq_front = temp;
    return;
}
temp2 = pq_front;
while(temp2->link->priority<temp->priority&&temp2->link!=NULL) {
    temp2 = temp2-> link;
}

temp-> link = temp2->link;
temp2-> link = temp;
}

void pq_delete() {
    struct pq * temp;
    if (front == NULL) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    temp = pq_front;
    printf("DELETED ELEMENT IS: %d", pq_front->data);
    pq_front = pq_front->link;
    free(temp);
}

void pq_display() {
    struct pq * temp;
    if (pq_front == NULL) {
        printf("QUEUE IS EMPTY!");
        return;
    }
    temp = pq_front;
    printf("QUEUE : ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
}
```



```
        break;
    case 4:
        exit(0);
    default:
        printf("WRONG CHOICE!");
    }
    getch();
}
case 2:
    while (1) {
        clrscr();
        printf("SIMPLE QUEUE DYNAMIC IMPLEMENTATION\n\n");
        printf("1. INSERT\n");
        printf("2. DELETE\n");
        printf("3. DISPLAY\n");
        printf("4. EXIT\n");
        printf("ENTER YOUR CHOICE : ");
        scanf("%d", & ch);
        switch (ch) {
            case 1:
                dynamic_insert();
                break;
            case 2:
                dynamic_delete();
                break;
            case 3:
                dynamic_display();
                break;
            case 4:
                exit(0);
            default:
                printf("WRONG CHOICE!");
        }
        getch();
    }
default:
    printf("WRONG CHOICE !");
}
getch();
}
case 2:
    while (1) {
        clrscr();
```

```
    printf("CIRCULAR QUEUE\n\n");
    printf("1. INSERT\n");
    printf("2. DELETE\n");
    printf("3. DISPLAY\n");
    printf("4. EXIT\n");
    printf("ENTER YOUR CHOICE : ");
    scanf("%d", & ch);
    switch (ch) {
    case 1:
        circular_insert();
        break;
    case 2:
        circular_delete();
        break;
    case 3:
        circular_display();
        break;
    case 4:
        exit(0);
    default:
        printf("WRONG CHOICE!");
    }
    getch();
}
case 3:
    while (1) {
        clrscr();
        printf("DOUBLE ENDED QUEUE\n\n");
        printf("1. INSERT\n");
        printf("2. DELETE\n");
        printf("3. DISPLAY\n");
        printf("4. EXIT\n");
        printf("ENTER YOUR CHOICE : ");
        scanf("%d", & ch);
        switch (ch) {
        case 1:
            double_insert();
            break;
        case 2:
            double_delete();
            break;
        case 3:
            double_display();
```

```
        break;
    case 4:
        exit(0);
    default:
        printf("WRONG CHOICE!");
    }
    getch();
}
case 4:
    while (1) {
        clrscr();
        printf("PRIORITY QUEUE\n\n");
        printf("1. INSERT\n");
        printf("2. DELETE\n");
        printf("3. DISPLAY\n");
        printf("4. EXIT\n");
        printf("ENTER YOUR CHOICE : ");
        scanf("%d", & ch);
        switch (ch) {
            case 1:
                pq_insert();
                break;
            case 2:
                pq_delete();
                break;
            case 3:
                pq_display();
                break;
            case 4:
                exit(0);
            default:
                printf("WRONG CHOICE!");
        }
        getch();
    }
case 5:
    exit(0);
default:
    printf("WRONG CHOICE!");
}
getch();
}
}
```

OUTPUT

```
QUEUE
```

```
1. SIMPLE QUEUE
2. CIRCULAR QUEUE
3. DOUBLE ENDED QUEUE
4. PRIORITY QUEUE
5. EXIT
ENTER YOUR CHOICE :
```

```
SIMPLE QUEUE
```

```
1. STATIC IMPLEMENTATION
2. DYANAMIC IMPLEMENTATION
ENTER YOUR CHOICE : _
```

```
SIMPLE QUEUE STATIC IMPLEMENTATION
```

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
ENTER YOUR CHOICE : 3
QUEUE : 12 56 90 67
```

CIRCULAR QUEUE

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
ENTER YOUR CHOICE : 3
QUEUE : 23 6 67 90 34 _
```

DOUBLE ENDED QUEUE

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
ENTER YOUR CHOICE : 1
1. INSERT AT FRONT END
2. INSERT AT REAR END
ENTER YOUR CHOICE : 1
ENTER DATA : 34_
```

DOUBLE ENDED QUEUE

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
ENTER YOUR CHOICE : 2
1. DELETE AT FRONT END
2. DELETE AT REAR END
ENTER YOUR CHOICE : 1
DELETE ELEMENT IS : 34_
```

PRIORITY QUEUE

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE : 1
ENTER DATA : 78
ENTER PRIORITY : 3

PRIORITY QUEUE

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE : 1
ENTER DATA : 2
ENTER PRIORITY : 1_

PRIORITY QUEUE

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE : 3
QUEUE : 2 78 _

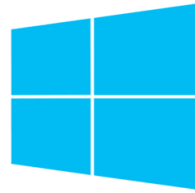


Linux

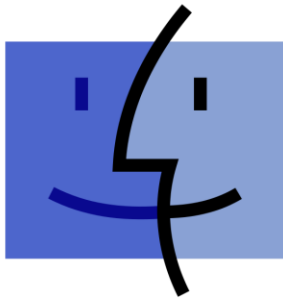
ubuntu[®]



ANDROID



Windows[®]



Mac OS